ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

AGENTIC LLM ΣΥΣΤΗΜΑ ΠΑΡΑΓΩΓΗΣ ΕΦΑΡΜΟΓΩΝ

Διπλωματική Εργασία

Κακκαβάς Παναγιώτης Κωνσταντίνος

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

AGENTIC LLM ΣΥΣΤΗΜΑ ΠΑΡΑΓΩΓΗΣ ΕΦΑΡΜΟΓΩΝ

Διπλωματική Εργασία

Παναγιώτης Κωνσταντίνος Κακκαβάς

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Οκτωβρίου 2025

……..	……..	……..
Π. Τσανάκας	Σ. Ξύδης	Χ. Παυλάτος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2025

Παναγιώτης Κωνσταντίνος Κακκαβάς
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στο πλαίσιο της παρούσας διπλωματικής, ο όρος «Appify» αναφέρεται στο σύστημα που αναπτύχθηκε και παρουσιάζεται, αναλύεται και αξιολογείται στο υπόλοιπο κείμενο.
Το Appify είναι μια καινοτόμος πλατφόρμα δημιουργίας εφαρμογών χωρίς κώδικα (no-code), που επιτρέπει σε οποιονδήποτε να μετατρέψει την ιδέα του σε λειτουργική εφαρμογή μέσα σε λίγα δευτερόλεπτα. Συνδυάζει ένα frontend σε Flutter/Dart (web, Android, iOS, desktop) με backend σε Python (FastAPI) που αξιοποιεί έξυπνους AI Agents (CrewAI) και αυτοματισμούς. Οι Appify AI Agents παράγουν αυτόματα πηγαίο κώδικα και επιτρέπουν με ένα κλικ πλήρες deployment της εφαρμογής σε όλες τις πλατφόρμες, από ιστοσελίδες και mobile apps έως desktop, smartwatches και in-car συστήματα. Έτσι, μια ιδέα γίνεται άμεσα διαθέσιμη στο κοινό χωρίς τεχνικές γνώσεις ή χρονοβόρες διαδικασίες.
Η χρήση του Appify είναι απλή: ο δημιουργός περιγράφει φυσικά την εφαρμογή που θέλει ή σχεδιάζει το UI της. Οι Agents αναλύουν την ιδέα, κατανοούν τις ανάγκες και παράγουν τον απαραίτητο κώδικα. Μέσα σε λίγα λεπτά, ο χρήστης αλληλεπιδρά με μια λειτουργική έκδοση και μπορεί να τη βελτιώνει με επιπλέον φυσικές περιγραφές. Το Appify προσφέρει επίσης αυτόματο deployment με ένα κλικ, επιτρέποντας τη φιλοξενία της εφαρμογής στους διακομιστές της πλατφόρμας ως web app.
Στόχος και όραμα: Το Appify στοχεύει να **δημοκρατικοποιήσει την ανάπτυξη εφαρμογών**, αφαιρώντας τα εμπόδια του προγραμματισμού και δίνοντας τη δύναμη δημιουργίας σε επιχειρηματίες, δημιουργούς, εκπαιδευτικούς και μικρές επιχειρήσεις. Επιτρέπει σε μη-προγραμματιστές να καινοτομήσουν σε χρόνους και κόστη αδιανόητα για τις παραδοσιακές μεθόδους. Σύμφωνα με μελέτες, τα εργαλεία no-code μπορούν να επιταχύνουν την ανάπτυξη έως και 10-20 φορές [1] [2]. Η φιλοσοφία του Appify συνοψίζεται στη φράση: **«μετατρέπει τα όνειρα σε εφαρμογές, γρήγορα, απλά και για όλους».**

Λέξεις Κλειδιά

no-code, multi-agent LLM, code generation, Flutter, CI/CD, DSL, App Stores, DevOps, MLOps, software synthesis

Abstract

Within the scope of this thesis, the term “Appify” refers to the developed system that is presented, analyzed, and evaluated throughout the document.

Appify is an innovative no-code application platform that enables any user to transform an idea into a working application within seconds. It is built on a hybrid stack of advanced technologies: a Flutter/Dart frontend for multi-platform apps (web, Android, iOS, desktop), combined with a robust Python (FastAPI) backend that leverages intelligent AI Agents (CrewAI) and automation tools. With the help of these Appify AI Agents, the system automatically generates high-quality source code and offers one-click full deployment across devices and platforms — from websites and mobile apps to desktop applications, smartwatches, and even in-vehicle systems. This means an idea can be turned into a publicly accessible application immediately, without requiring programming expertise or lengthy development cycles.

Using Appify is simple and natural. Creators begin by describing in natural language the application they want—or by sketching a rough UI. This description may concern a brand-new project or an enhancement to an existing app. The platform’s intelligent agents analyze the idea, understand its context and usage requirements, and then produce the necessary code. Within seconds, the user can interact with the live application generated by the platform, and continue evolving it by adding new features and improvements through additional natural-language prompts. Appify does not stop at creation — it also provides full one-click deployment. The user can host the app on the platform’s servers (as a web app), without complex procedures. In this way, any idea can be realized and reach its intended audience quickly.

Vision and Goal: Appify aims to democratize application development. By removing traditional programming barriers, it empowers entrepreneurs, content creators, educators, small businesses, and anyone with an idea to turn it into a product. It is a platform that enables non-programmers to innovate with timeframes and costs previously unattainable via conventional development methods. According to studies, no-code tools can accelerate software development by up to 10× and require ~70% fewer resources than traditional development. In line with these trends, Appify’s philosophy is to “turn dreams into apps — fast, simple, and for everyone.”

Keywords

no-code, multi-agent LLM, code generation, Flutter, CI/CD, DSL, App Stores, DevOps, MLOps, software synthesis

Ευχαριστίες

Με την ολοκλήρωση αυτής της διπλωματικής εργασίας, θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη προς όλους εκείνους που με στήριξαν και συνέβαλαν σε αυτήν την πορεία.

Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον κύριο Π. Τσανάκα για την πολύτιμη καθοδήγηση, την αδιάκοπη υποστήριξη και την ουσιαστική συμβολή του σε κάθε στάδιο της εργασίας μου. Οι συμβουλές του και οι τακτικές συναντήσεις μας προκειμένου να παρακολουθεί και να καθοδηγεί την πορεία της εργασίας μου, αποτέλεσαν ακρογωνιαίο λίθο για την επιτυχή ολοκλήρωση αυτής.

Ιδιαίτερες ευχαριστίες οφείλω επίσης τους διδακτορικούς φοιτητές, Γ. Μπρατσιώτης, και Κ. Αθανασιάδη για τη συνεχή βοήθεια και την εξαιρετική συνεργασία που είχαμε καθ' όλη τη διάρκεια αυτής της διαδικασίας. Η συνεισφορά του ήταν ανεκτίμητη και η καθοδήγησή του καίρια.

Ευχαριστώ θερμά και τους καθηγητές μου, τον κύριο Χ. Παυλάτος και τον κύριο Σ. Ξύδης**,** για τη συμμετοχή τους στην τριμελή επιτροπή και για το χρόνο που αφιέρωσαν στην αξιολόγηση της εργασίας μου.

Επιπρόσθετα, ευχαριστίες αξίζουν στους φίλους και τη σύντροφό μου, που με στήριξαν, με ενθάρρυναν και μου προσέφεραν την αγάπη, την παρέα και τη φιλία τους καθ' όλη τη διάρκεια αυτής της διαδρομής. Η παρουσία τους υπήρξε ανεκτίμητη και με βοήθησε να ανταπεξέλθω σε κάθε δυσκολία.

Τις βαθύτερες ευχαριστίες και την απέραντη ευγνωμοσύνη μου αξίζει η οικογένειά μου, για τη συνεχή τους στήριξη, την ακούραστη ενθάρρυνση και την αμέριστη αγάπη τους. Η πίστη τους σε μένα και η ηθική και υλική τους υποστήριξη ήταν οι πυλώνες που με κράτησαν σταθερό σε αυτή την πορεία.

Σας ευχαριστώ όλους από καρδιάς.

Περιεχόμενα

Πίνακας Εικόνων

Λίστα Ακρωνυμίων

TTFP:	Time To First Prototype
API:	Application Programming Interface
IDE:	Integrated Development Environment
UI:	User Interface
JSON:	JavaScript Object Notation
HTTP(S):	Hypertext Transfer Protocol (Secure)
REST:	Representational State Transfer

Κεφάλαιο 1: Εισαγωγή, Ιστορικό και Κίνητρα

1.1 Το πρόβλημα

Η ραγδαία εξάπλωση των ψηφιακών υπηρεσιών και των έξυπνων συσκευών έχει καταστήσει τη δημιουργία εφαρμογών αναπόσπαστο κομμάτι της καθημερινότητας για επιχειρήσεις, εκπαιδευτικά ιδρύματα και ανεξάρτητους δημιουργούς [3]. Ωστόσο, η **παραδοσιακή ανάπτυξη λογισμικού** εξακολουθεί να αποτελεί μια σύνθετη και χρονοβόρα διαδικασία που απαιτεί εξειδικευμένες γνώσεις προγραμματισμού, εμπειρία σε διαφορετικά περιβάλλοντα ανάπτυξης (IDE), καθώς και κατανόηση τεχνολογιών όπως βάσεις δεδομένων, APIs, συστήματα ελέγχου εκδόσεων και διαδικασίες διανομής (app stores) [4].

Η πολυπλοκότητα αυτή δημιουργεί ένα σημαντικό **χάσμα μεταξύ της σύλληψης μιας ιδέας και της πραγματικής υλοποίησής της**, το οποίο συχνά οδηγεί στη ματαίωση καινοτόμων εγχειρημάτων. Ιδιαίτερα για άτομα ή μικρές ομάδες χωρίς τεχνικό υπόβαθρο —όπως εκπαιδευτικούς, ερευνητές, μικρομεσαίους επιχειρηματίες ή δημιουργούς περιεχομένου— η είσοδος στον χώρο ανάπτυξης εφαρμογών παραμένει δυσπρόσιτη.

Παράλληλα, ακόμη και για επαγγελματικές ομάδες ανάπτυξης, ο χρόνος που μεσολαβεί από τη σύλληψη της ιδέας έως την παραγωγή ενός πρώτου λειτουργικού πρωτοτύπου (**Time-to-First-Prototype, TTFP**) μπορεί να είναι αρκετές εβδομάδες ή και μήνες. Η ύπαρξη πολλαπλών ρόλων (σχεδιαστές, προγραμματιστές, DevOps) και εργαλείων καθυστερεί τον κύκλο ανάπτυξης και αυξάνει το κόστος. Το αποτέλεσμα είναι ότι πολλές ιδέες δεν υλοποιούνται ποτέ ή φτάνουν στην αγορά με καθυστέρηση, όταν οι συνθήκες έχουν ήδη αλλάξει [1].

Εικόνα 1 Γενική Διαδικασία Ανάπτυξης Εφαρμογής από Ιδέα σε Προϊον [1]

Το πρόβλημα αυτό αποτελεί το σημείο εκκίνησης για το Appify, μια πλατφόρμα που επιχειρεί να εξαλείψει τα παραπάνω εμπόδια, καθιστώντας δυνατή τη δημιουργία πραγματικών, λειτουργικών εφαρμογών χωρίς την ανάγκη γνώσης προγραμματισμού.

1.2 Σημασία της αμεσότητας στη δημιουργία εφαρμογών

Στο σημερινό τεχνολογικό περιβάλλον, η **ταχύτητα με την οποία μια ιδέα μπορεί να μετατραπεί σε προϊόν** είναι συχνά πιο κρίσιμη από την ίδια την ιδέα. [5] Οι επιχειρήσεις και οι δημιουργοί χρειάζονται τρόπους να επαληθεύουν τις υποθέσεις τους γρήγορα, να δοκιμάζουν λειτουργίες και να συλλέγουν ανατροφοδότηση από τους χρήστες όσο το δυνατόν νωρίτερα.

Η παραδοσιακή διαδικασία ανάπτυξης καθιστά αυτόν τον στόχο δύσκολο. [6] Οι ροές ανάπτυξης περιλαμβάνουν πολλαπλά βήματα: σχεδιασμό, προγραμματισμό, δοκιμές, δημιουργία builds, διανομή, και επαναληπτικό debugging. Κάθε ένα από αυτά τα στάδια απαιτεί διαφορετικά εργαλεία και ικανότητες. Η ανάγκη για συντονισμό μεταξύ διαφορετικών ρόλων αυξάνει περαιτέρω την πολυπλοκότητα.

Η **αμεσότητα** (immediacy ή time-to-value) στη δημιουργία εφαρμογών μεταφράζεται σε μείωση του χρόνου που χρειάζεται ένας χρήστης για να δει το αποτέλεσμα της ιδέας του σε λειτουργική μορφή. Αυτό αποτελεί καθοριστικό παράγοντα επιτυχίας για κάθε σύγχρονη πλατφόρμα no-code/low-code. [7]

Το **Appify** στοχεύει στη ριζική μείωση του χρόνου από την ιδέα έως το πρωτότυπο: από εβδομάδες σε λεπτά ή ώρες, αξιοποιώντας την ισχύ των **μεγάλων γλωσσικών μοντέλων (LLMs)** και τη συνεργασία **ευφυών πρακτόρων (Appify AI Agents)**. [8] [9] Μέσω φυσικής γλώσσας ή απλών σχεδιαστικών περιγραφών, ο χρήστης μπορεί να δημιουργήσει ένα πλήρως λειτουργικό έργο που να εκτελείται στο web, σε κινητές συσκευές ή στον υπολογιστή του.

Η έμφαση στην αμεσότητα δεν αφορά μόνο την ταχύτητα, αλλά και την **ποιότητα της εμπειρίας:** ο χρήστης δεν περιμένει παθητικά το αποτέλεσμα, αλλά παρακολουθεί την εφαρμογή να «χτίζεται» μπροστά του, με **hot reload σε πραγματικό χρόνο** και διαδραστική επικοινωνία με τους πράκτορες. [10] Αυτή η προσέγγιση ενισχύει τη δημιουργικότητα, μειώνει την απόσταση μεταξύ ιδέας και εκτέλεσης, και καθιστά τη διαδικασία ανάπτυξης προσβάσιμη σε όλους.

1.3 Στόχος της εφαρμογής

Η παρούσα διπλωματική εργασία έχει ως στόχο τη **σχεδίαση και υλοποίηση** μιας πλήρως λειτουργικής πλατφόρμας **no-code ανάπτυξης εφαρμογών με υποστήριξη πρακτόρων τεχνητής νοημοσύνης (AI Agents).** [11]

Συγκεκριμένα, το **Appify** αποσκοπεί:

1. Στη **συλλογή και κατανόηση απαιτήσεων** από τον χρήστη μέσω φυσικής γλώσσας (prompt) ή σχεδίου διεπαφής. [12]
2. Στη **σύνθεση τεχνικών προδιαγραφών** (UI/UX, οντοτήτων δεδομένων, ροών λειτουργιών).
3. Στην **παραγωγή κώδικα** (code generation) σε γλώσσα **Dart/Flutter**, με συνοδευτικό backend όπου χρειάζεται.
4. Στην **αυτόματη εκτέλεση διαδικασιών build, test και deployment** μέσα σε απομονωμένο περιβάλλον Docker. [13] [14]

5. Στην **παρουσίαση και διάθεση** του αποτελέσματος στον χρήστη μέσω ενός περιβάλλοντος προεπισκόπησης (preview) ή έτοιμου προς δημοσίευση app.

Η διαδικασία αυτή επιτυγχάνεται μέσω της συνεργασίας διαφορετικών τύπων πρακτόρων, όπως:

1. **Idea Validation Agent**, που ελέγχει τη συνέπεια και υλοποιησιμότητα της ιδέας.
2. **Architect Agent**, που μεταφράζει την ιδέα σε σχεδιασμό εφαρμογής.
3. **Developer Agent**, που παράγει, τροποποιεί και δοκιμάζει κώδικα. [15]
4. **Critic Agent**, που αξιολογεί τα αποτελέσματα και προτείνει διορθώσεις.

Η υλοποίηση συνδυάζει τεχνολογίες αιχμής: **CrewAI** για συντονισμό πολυπρακτορικών ροών, **FastAPI** για τη διαχείριση αιτημάτων, **Docker** και **tmux** για απομονωμένη εκτέλεση build, **Flutter** για την παραγωγή πολυπλατφορμικού UI, καθώς και **LiteLLM** και **Serper.dev** για επικοινωνία με διαφορετικά LLMs και αναζήτηση πληροφορίας.

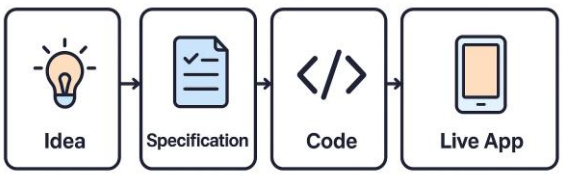
Η πλατφόρμα δοκιμάστηκε πειραματικά σε σενάρια δημιουργίας εφαρμογών, αξιολογώντας δείκτες όπως:

- μέσος χρόνος παραγωγής πρώτου πρωτοτύπου,
- μέσος αριθμός tokens ανά αρχείο Dart (mean 338.5, std 230.2),
- ικανότητα αυτοδιόρθωσης μέσω feedback prompts.

1.4 Καινοτομία

Η καινοτομία του Appify εντοπίζεται στον **συνδυασμό τεχνητής νοημοσύνης, αυτοματοποίησης και παραδοσιακής ανάπτυξης λογισμικού** σε ένα ενιαίο, αλληλεπιδραστικό περιβάλλον. [16]

Σε αντίθεση με τις υπάρχουσες no-code πλατφόρμες που βασίζονται σε έτοιμα blocks ή απλές “φόρμες”, το Appify **παράγει πραγματικό πηγαίο κώδικα,** ο οποίος μπορεί να επιθεωρηθεί, να επεκταθεί ή να εξαχθεί εκτός πλατφόρμας. Η παραγωγή αυτή γίνεται μέσω πρακτόρων που λειτουργούν με τη μέθοδο **REACT (Reason + Act)**, η οποία βασίζεται στη διαδοχή Thought → Action → Observation → Answer. Οι πράκτορες, αντί να απαντούν απλώς με κείμενο, εκτελούν ενέργειες: γράφουν αρχεία, τρέχουν εντολές, πραγματοποιούν αναζητήσεις, ελέγχουν αποτελέσματα και διορθώνουν τον κώδικα σε πραγματικό χρόνο. [17] [18]



Εικόνα 2User Interface Workflow

Επιπλέον, το Appify ενσωματώνει μια **αυτόνομη υποδομή build** με χρήση **Docker containers και tmux**, επιτρέποντας στο Flutter project να εκτελείται και να ανανεώνεται (hot reload) μέσα σε απομονωμένο περιβάλλον. Αυτό καθιστά τη διαδικασία ανάπτυξης **πλήρως αυτοματοποιημένη και αναπαραγώγησιμη**, ενώ παρέχει οπτική ανατροφοδότηση στον χρήστη.

Η προσέγγιση του Appify συνιστά ένα νέο παράδειγμα στην ανάπτυξη λογισμικού: **ο χρήστης μετατρέπεται από προγραμματιστής σε σκηνοθέτης διαδικασιών**, και το σύστημα αναλαμβάνει τον ρόλο της εκτέλεσης.
Μέσω αυτής της αλληλεπίδρασης ανθρώπου-μηχανής, επιτυγχάνεται όχι μόνο ταχύτητα, αλλά και **δημοκρατικοποίηση της δημιουργίας εφαρμογών** — ένα από τα βασικά οράματα της σύγχρονης τεχνητής νοημοσύνης.

Συνοψίζοντας, το Appify εισάγει μια νέα προσέγγιση στην ανάπτυξη λογισμικού, γεφυρώνοντας το χάσμα μεταξύ φυσικής γλώσσας και προγραμματισμού.
Με τη χρήση πρακτόρων τεχνητής νοημοσύνης, την ενοποίηση σύγχρονων εργαλείων ανάπτυξης και την άμεση διάθεση εφαρμογών, **συνδυάζει τεχνική καινοτομία με ανθρώπινη δημιουργικότητα**.
Το επόμενο κεφάλαιο παρουσιάζει αναλυτικά το τεχνολογικό υπόβαθρο, τις πλατφόρμες και τα εργαλεία πάνω στα οποία στηρίχθηκε η υλοποίηση της πλατφόρμας Appify.

Κεφάλαιο 2: ΤΕΧΝΟΛΟΓΙΚΟ ΥΠΟΒΑΘΡΟ ΚΑΙ ΕΡΓΑΛΕΙΑ ΑΝΑΠΤΥΞΗΣ

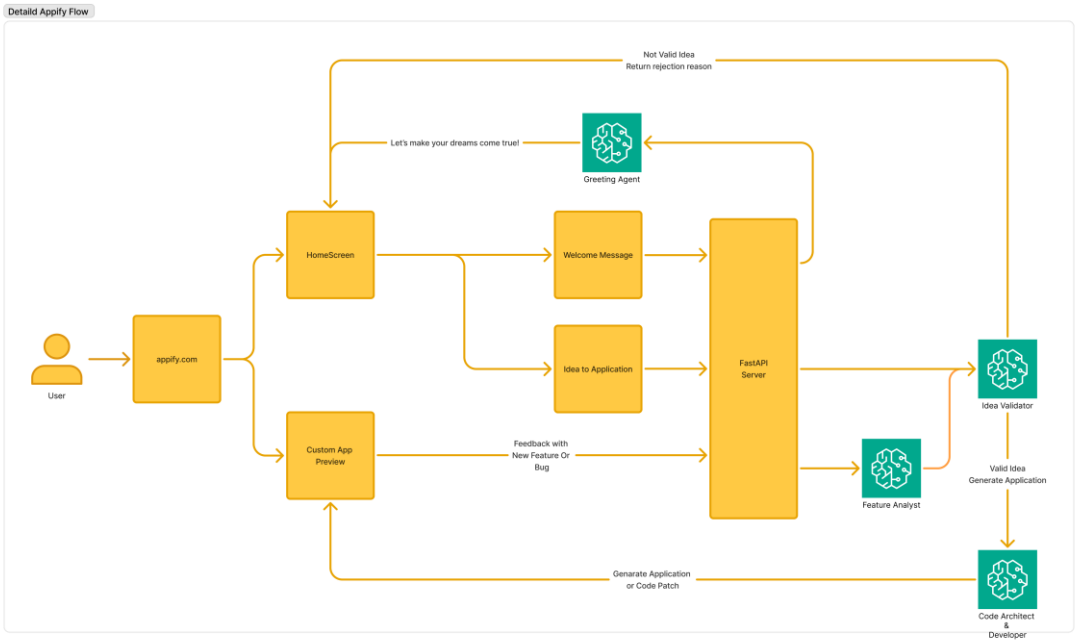
2.1 Επισκόπηση Αρχιτεκτονικής Appify

2.1.1 Γενικά

Η πλατφόρμα Appify σχεδιάστηκε ως ένα πολυεπίπεδο σύστημα που συνδυάζει τεχνολογίες ανάπτυξης εφαρμογών, τεχνητής νοημοσύνης και αυτοματοποίησης, με σκοπό τη δημιουργία ενός πλήρως λειτουργικού περιβάλλοντος no-code. Ο στόχος της αρχιτεκτονικής είναι να επιτρέπει τη μετατροπή φυσικής γλώσσας ή γραφικού σχεδίου σε λειτουργικό κώδικα (code synthesis) και στη συνέχεια το build και deploy της εφαρμογής με ελάχιστη ανθρώπινη παρέμβαση. [19]

Η αρχιτεκτονική του Appify είναι υβριδική, συνδυάζοντας στοιχεία frontend, backend και DevOps σε ένα ενιαίο οικοσύστημα. [20] Ο πυρήνας της βασίζεται σε ένα frontend γραμμένο σε Flutter/Dart, το οποίο επιτρέπει την παραγωγή εφαρμογών για πολλαπλές πλατφόρμες (web, Android, iOS, desktop). [21] Το frontend επικοινωνεί με ένα backend σε Python, το οποίο λειτουργεί ως “κινητήρας νοημοσύνης” και συντονίζει τη λειτουργία των AI πρακτόρων (Appify AI Agents). Αυτοί οι πράκτορες αξιοποιούν μεγάλα γλωσσικά μοντέλα (LLMs) για την ανάλυση των απαιτήσεων, τη δημιουργία και τον έλεγχο κώδικα, καθώς και για την αυτοματοποίηση ενεργειών μέσα σε ένα Dockerized περιβάλλον ανάπτυξης.

Η επικοινωνία μεταξύ των επιπέδων γίνεται μέσω RESTful APIs (FastAPI framework) και εσωτερικών μηχανισμών ανταλλαγής μηνυμάτων. [22] [23] Η παραγωγή του κώδικα γίνεται μέσα στο container του Flutter, ενώ οι πράκτορες αλληλεπιδρούν μέσω ενός μηχανισμού tmux sessions, που επιτρέπει αποστολή εντολών και εκτέλεση αναλύσεων σε πραγματικό χρόνο.



Εικόνα 3 Αναλυτικό Appify Workflow

2.1.2 Διάγραμμα Υψηλού Επιπέδου

Η συνολική αρχιτεκτονική του Appify αποτελείται από πέντε βασικά υποσυστήματα:

1. User Interface Layer: περιλαμβάνει το περιβάλλον αλληλεπίδρασης με τον χρήστη, όπου δίνεται η περιγραφή ή το σχέδιο της εφαρμογής.
2. AI Processing Layer: το επίπεδο που υλοποιεί τη νοημοσύνη των πρακτόρων, με συντονισμό μέσω του framework CrewAI.
3. Code Generation Layer: υπεύθυνο για τη σύνθεση του Dart/Flutter κώδικα και τη διαχείριση αρχείων έργου.
4. Build & Deployment Layer: το containerized περιβάλλον που αναλαμβάνει το build, testing και hosting της εφαρμογής. [24]
5. Monitoring & Telemetry Layer: καταγράφει μεταδεδομένα, μετρήσεις απόδοσης και logs για αξιολόγηση και βελτιστοποίηση. [25]

2.1.3 Ροή Επικοινωνίας

Η αλληλεπίδραση μεταξύ των υποσυστημάτων πραγματοποιείται σε διαδοχικά στάδια:

1. Ο χρήστης αποστέλλει ένα prompt ή σχέδιο εφαρμογής στο backend μέσω FastAPI.
2. Ο AI Orchestrator Agent αναλαμβάνει να εκκινήσει μια ροή (flow) πρακτόρων: Idea Validation → Brainstorming → Architecture → Development.
3. Κατά την υλοποίηση, οι πράκτορες γράφουν κώδικα απευθείας στο φάκελο του έργου (μέσω εργαλείων filesystem) και ενεργοποιούν builds και hot reload μέσα στο Docker container.
4. Το frontend (που τρέχει στον browser μέσω του Flutter web server) ενημερώνεται σε πραγματικό χρόνο, επιτρέποντας στον χρήστη να βλέπει την εφαρμογή του να “γεννιέται” δυναμικά.
5. Τέλος, τα αποτελέσματα αποθηκεύονται, γίνονται commit και versioning στο Git repository και μπορούν να διατεθούν σε παραγωγή.

2.1.4 Πλεονεκτήματα Υβριδικής Αρχιτεκτονικής

Η υβριδική προσέγγιση της πλατφόρμας προσφέρει πολλαπλά πλεονεκτήματα:

- **Απομόνωση επιπέδων:** κάθε υποσύστημα (UI, AI, Build) μπορεί να αναπτυχθεί ή αναβαθμιστεί ανεξάρτητα.
- **Επαναληψιμότητα:** μέσω containers και tmux sessions, η διαδικασία παραγωγής κώδικα είναι αναπαραγώγιμη και αυτοματοποιημένη.
- **Επεκτασιμότητα:** το CrewAI framework επιτρέπει προσθήκη νέων πρακτόρων χωρίς αλλαγές στον κορμό.
- **Διαφάνεια:** ο χρήστης μπορεί να δει τον παραγόμενο κώδικα, διατηρώντας τον πλήρη έλεγχο της εφαρμογής του.

2.2 Το Πλαίσιο Ανάπτυξης Flutter

2.2.1 Εισαγωγή στο Flutter

Το **Flutter** αποτελεί ένα **ανοιχτού κώδικα πλαίσιο ανάπτυξης διεπαφών χρήστη (UI Software Development Kit)** που αναπτύχθηκε από τη **Google** και πρωτοπαρουσιάστηκε το 2015, με την πρώτη του σταθερή έκδοση να κυκλοφορεί το 2017. Επιτρέπει τη **δημιουργία εφαρμογών πολλαπλών πλατφορμών (cross-platform)** μέσω ενός ενιαίου κώδικα, ο οποίος μπορεί να εκτελείται σε **Android, iOS, Web, Windows, macOS, Linux** και, σε πειραματικό στάδιο, στο λειτουργικό **Fuchsia** της Google [26].

Το Flutter χρησιμοποιείται τόσο εσωτερικά από τη Google (σε εφαρμογές όπως **Google Pay** και **Google Earth**) όσο και από κορυφαίες εταιρείες όπως η **ByteDance** και η **Alibaba**, επιβεβαιώνοντας τη σταθερότητα και την παραγωγική του ωριμότητα. Κύριο χαρακτηριστικό του είναι η ικανότητά του να προσφέρει **εμπειρία χρήστη συγκρίσιμη με αυτή των εγγενών εφαρμογών (native apps)**, παρέχοντας παράλληλα ευελιξία, ταχύτητα ανάπτυξης και χαμηλότερο κόστος συντήρησης [27].

2.2.2 Βασικά Χαρακτηριστικά και Πλεονεκτήματα

Το Flutter συνδυάζει τεχνικά χαρακτηριστικά που το καθιστούν ένα από τα πιο παραγωγικά και ευέλικτα περιβάλλοντα ανάπτυξης της τελευταίας δεκαετίας:

- **Ενιαίος Κώδικας για Πολλαπλές Πλατφόρμες:**
  Οι εφαρμογές αναπτύσσονται μία φορά και μπορούν να εκτελεστούν σε διαφορετικά λειτουργικά συστήματα χωρίς ουσιαστικές αλλαγές στον πηγαίο κώδικα. Αυτό μειώνει δραστικά τον χρόνο και το κόστος ανάπτυξης, διευκολύνοντας παράλληλα τη συντήρηση.
- **Υψηλή Απόδοση:**
  Η μηχανή απεικόνισης (**Skia Rendering Engine**) του Flutter, γραμμένη σε C++, επιτρέπει τη δημιουργία γραφικών με ταχύτητα και ακρίβεια, αποδίδοντας εμπειρία χρήστη αντίστοιχη των native εφαρμογών.
- **Πλούσια Βιβλιοθήκη Widgets:**
  Το Flutter διαθέτει ένα εκτενές σύνολο **προσαρμόσιμων widgets**, τα οποία αποτελούν τη βασική μονάδα δόμησης των διεπαφών χρήστη. Κάθε εφαρμογή Flutter αποτελείται από ένα **δέντρο widgets (widget tree)** που περιγράφει τη λογική, τη διάταξη και την αλληλεπίδραση του UI. [28]
- **Δηλωτικός και Αντιδραστικός Προγραμματισμός (Reactive Programming):**
  Το Flutter υποστηρίζει δηλωτικό μοντέλο ανάπτυξης, όπου το UI ενημερώνεται αυτόματα όταν αλλάζουν τα δεδομένα. Αυτή η προσέγγιση απλοποιεί σημαντικά τη διαχείριση κατάστασης (state management).

- **Άμεση Ανανέωση (Hot Reload):**
  Οι αλλαγές στον κώδικα εμφανίζονται άμεσα χωρίς επανεκκίνηση της εφαρμογής, γεγονός που επιταχύνει τη διαδικασία ανάπτυξης και δοκιμών. [10]
- **Πλούσιο Οικοσύστημα και Ενεργή Κοινότητα:**
  Το Flutter υποστηρίζεται από μία από τις πιο δραστήριες κοινότητες ανοιχτού κώδικα, με χιλιάδες βιβλιοθήκες και πακέτα που επεκτείνουν τις δυνατότητές του.

2.2.3 Αρχιτεκτονική Εφαρμογής Flutter

Κάθε έργο Flutter περιλαμβάνει έναν **κύριο φάκελο** lib/, όπου αποθηκεύεται ο πηγαίος κώδικας σε Dart, καθώς και επιμέρους φακέλους για κάθε υποστηριζόμενη πλατφόρμα (android/, ios/, web/, macos/, linux/, windows/). Ο **αρχικός κώδικας της εφαρμογής** βρίσκεται στο αρχείο lib/main.dart, το οποίο εκκινεί την εφαρμογή και διαχειρίζεται το σύστημα πλοήγησης (navigation). [29]

Οι πράκτορες του Appify δημιουργούν αυτόματα νέα αρχεία και φακέλους, όπως lib/screens/ για τις οθόνες, lib/widgets/ για τα επαναχρησιμοποιήσιμα γραφικά στοιχεία και lib/services/ για τη λογική της εφαρμογής. Αυτή η δομή διασφαλίζει καθαρό διαχωρισμό ευθυνών (separation of concerns) και διευκολύνει την αυτοματοποίηση παραγωγής κώδικα από τους AI πράκτορες.

```
xapp
├── README.md
├── analysis_options.yaml
├── android
├── assets
├── build
├── ios
├── lib
├── pubspec.lock
├── pubspec.yaml
├── test
├── web
├── xapp
├── xapp.iml
└── xapp.log

8 directories, 6 files
```

Εικόνα 4 Δομή Flutter Εφαρμογής

```
xapp/lib/
├── main.dart
├── models
│   ├── detection.dart
│   ├── health_status.dart
│   ├── predict_request.dart
│   └── predict_response.dart
├── painters
│   └── bounding_boxes_painter.dart
├── repository
│   └── detection_repository.dart
├── screens
│   └── detection_page.dart
├── services
│   ├── api_client.dart
│   ├── api_config.dart
│   └── config_service.dart
├── utils
│   ├── feedback_button.dart
│   └── geometry.dart
└── widgets
    └── drop_image_zone.dart

7 directories, 14 files
```

Εικόνα 5 Δομή lib της Flutter Εφαρμογής

2.2.4 Υποστηριζόμενες Πλατφόρμες

Το Flutter επιτρέπει ενιαίο codebase για Web, Android, iOS και Desktop εφαρμογές. Στο Appify, οι πράκτορες μπορούν να καθορίσουν τη στοχευόμενη πλατφόρμα build, χρησιμοποιώντας εντολές όπως: [30] [31] [32]

```
1. flutter build web
2. flutter build apk
3. flutter build macos
```

Αυτή η δυνατότητα επιτρέπει τη γρήγορη διάθεση εφαρμογών σε πολλαπλά περιβάλλοντα, μειώνοντας την ανάγκη για ξεχωριστές ομάδες ανάπτυξης ανά πλατφόρμα.

2.2.5 Πλεονεκτήματα και Μειονεκτήματα

Πλεονεκτήματα:

- Ταχύτητα ανάπτυξης και ενιαία εμπειρία χρήστη σε όλες τις πλατφόρμες.
- Πλούσια βιβλιοθήκη widgets και εκτεταμένες δυνατότητες προσαρμογής.
- Ευκολία ενσωμάτωσης **AI πρακτόρων**, καθώς ο Dart κώδικας είναι απλός στην ανάλυση και τη δημιουργία.
- Υψηλή απόδοση, κοντά σε native επίπεδο.

Μειονεκτήματα:

- **Μεγαλύτερο μέγεθος τελικού build** σε σχέση με εγγενείς εφαρμογές.

- **Περιορισμένη native διαλειτουργικότητα** για πολύπλοκες λειτουργίες χωρίς τη χρήση εξωτερικών plugins.
- **Μικρότερη ωριμότητα** της υποστήριξης για web και desktop σε σχέση με mobile πλατφόρμες.

2.2.6 Χρήση στο Appify

Στο πλαίσιο του Appify, το Flutter χρησιμοποιείται τόσο για την **frontend διεπαφή του χρήστη** (μέσω της οποίας ο χρήστης αλληλεπιδρά με το σύστημα και παρέχει prompts), όσο και ως **βάση για την παραγωγή του τελικού προϊόντος** - της αυτόματα δημιουργούμενης εφαρμογής.

Η διαδικασία ανάπτυξης πραγματοποιείται εντός **Docker container**, το οποίο εκκινεί **tmux session** με την εντολή:

```
1. flutter run -d web-server --web-port=8080
```

Με τον τρόπο αυτό, κάθε αλλαγή που παράγεται από τους πράκτορες ενεργοποιεί **αυτόματο hot reload** στην εφαρμογή, επιτρέποντας στο χρήστη να βλέπει άμεσα το αποτέλεσμα των ενεργειών του. Το σύστημα αυτό καθιστά το Appify ικανό να λειτουργεί ως **πραγματικός "ζωντανός" δημιουργός εφαρμογών**, παρέχοντας στον χρήστη μια εμπειρία ανάπτυξης που συνδυάζει αυτοματοποίηση, αμεσότητα και διαδραστικότητα.

2.3 Η Γλώσσα Προγραμματισμού Dart

2.3.1 Γενικά

Η Dart δημιουργήθηκε από τη Google το 2011 με στόχο να προσφέρει μια εναλλακτική στη JavaScript για client-side προγραμματισμό. Από το 2018 έγινε η βασική γλώσσα του Flutter. [33]

2.3.2 Συντακτικά Χαρακτηριστικά

Η Dart είναι **strongly typed**, υποστηρίζει **null safety, asynchronous προγραμματισμό** (async/await) και **object-oriented** σύνταξη. Αυτά τα χαρακτηριστικά καθιστούν τον κώδικα που παράγεται από τους πράκτορες πιο σταθερό και ευανάγνωστο.

2.3.3 Dart SDK και Toolchain

Η **Dart** είναι μια σύγχρονη γλώσσα προγραμματισμού που αναπτύχθηκε από τη Google και σχεδιάστηκε από τους **Lars Bak** και **Kasper Lund**, με σκοπό να προσφέρει ένα ισχυρό, αποδοτικό και ευέλικτο υπόβαθρο για την ανάπτυξη εφαρμογών πολλαπλών πλατφορμών. Η Dart χρησιμοποιείται εκτενώς στο οικοσύστημα του **Flutter**, καθώς παρέχει έναν βελτιστοποιημένο μηχανισμό για τη δημιουργία

διεπαφών χρήστη (UI) υψηλής απόδοσης και φυσικής αίσθησης, τόσο για κινητές συσκευές όσο και για τον ιστό και desktop εφαρμογές.

Πρόκειται για μια **αντικειμενοστραφή, βασισμένη σε κλάσεις** γλώσσα με **ισχυρή τυποποίηση (strong typing), συλλογή απορριμμάτων (garbage collection)** και **σύνταξη εμπνευσμένη από τη C# και τη Java**. Υποστηρίζει **διεπαφές (interfaces)**, **αφηρημένες κλάσεις, μίξεις (mixins), γενικεύσεις (generics)** και **κληρονομικότητα (inheritance)**, επιτρέποντας τη δημιουργία αναγνώσιμου, επαναχρησιμοποιήσιμου και καλά οργανωμένου κώδικα.

Η Dart προσφέρει επίσης **προηγμένες δυνατότητες ασύγχρονου προγραμματισμού**, μέσω των δομών **Futures** και **Streams**, επιτρέποντας την εκτέλεση χρονοβόρων λειτουργιών (όπως κλήσεις δικτύου ή ανάγνωση αρχείων) χωρίς να μπλοκάρεται η κύρια ροή της εφαρμογής. Το συντακτικό της υιοθετεί το μοντέλο **async/await**, παρόμοιο με αυτό της JavaScript, αλλά με αυστηρότερη τυποποίηση και καλύτερη διαχείριση σφαλμάτων.

Ένα από τα σημαντικότερα χαρακτηριστικά της Dart είναι η **ευελιξία στη μεταγλώττιση**: μπορεί να μεταγλωττιστεί τόσο σε **εγγενή κώδικα (ARM, x86)** όσο και σε **JavaScript ή WebAssembly**, καθιστώντας δυνατή την εκτέλεση των εφαρμογών της σχεδόν σε οποιαδήποτε συσκευή ή πρόγραμμα περιήγησης. Αυτή η ιδιότητα καθιστά τη Dart μια πραγματικά **πολυπλατφορμική γλώσσα**, ιδανική για έργα όπως το Appify, τα οποία απαιτούν παραγωγή εφαρμογών σε πολλαπλά περιβάλλοντα χωρίς αλλαγή στον πυρήνα του κώδικα. [34] [35]

Το **Dart SDK** αποτελεί τον πυρήνα του εργαλειακού οικοσυστήματος της γλώσσας και περιλαμβάνει τον **μεταγλωττιστή (compiler)**, το εργαλείο διαχείρισης πακέτων pub, καθώς και εντολές για ανάλυση, έλεγχο και μορφοποίηση κώδικα (dart analyze, dart test, dart format). Σε συνδυασμό με το **Flutter SDK**, παρέχεται ένα πλήρες **toolchain ανάπτυξης**, το οποίο περιλαμβάνει εντολές όπως flutter doctor, flutter pub, flutter build και flutter run, διασφαλίζοντας τη δημιουργία, τον έλεγχο, την ανάλυση και το deployment των εφαρμογών.

Η **στατική ανάλυση** κώδικα και η **διαχείριση εξαρτήσεων** πραγματοποιούνται μέσω του αρχείου pubspec.yaml, το οποίο περιγράφει τις βιβλιοθήκες, τα assets και τις ρυθμίσεις build. Με αυτόν τον τρόπο διασφαλίζεται η **σταθερότητα και η συνέπεια** του παραγόμενου κώδικα. Παράλληλα, τα ενσωματωμένα εργαλεία του IDE, όπως το **Flutter Inspector**, το **Outline View**, και το **DevTools**, παρέχουν δυνατότητες **αποσφαλμάτωσης (debugging)**, **ανάλυσης επιδόσεων (profiling)** και **παρακολούθησης μνήμης/CPU**. [36]

Ο Flutter Inspector επιτρέπει την **επιθεώρηση του δέντρου widgets (widget tree)** και τη ζωντανή τροποποίηση των ιδιοτήτων των στοιχείων διεπαφής, ενώ το DevTools προσφέρει **ανάλυση καρέ (frame timeline)** και **προφίλ απόδοσης**. Μέσω αυτού του οικοσυστήματος εργαλείων, το Appify μπορεί **να παράγει, να επιθεωρεί και να αξιολογεί αυτόματα τον κώδικα** που δημιουργούν οι πράκτορές του, εξασφαλίζοντας υψηλή ποιότητα και συμμόρφωση στις προδιαγραφές UI/UX. [37]

Συνολικά, η Dart, σε συνδυασμό με το Flutter SDK, συνιστά μια **συνεκτική και ώριμη πλατφόρμα ανάπτυξης** που ευνοεί την αυτοματοποίηση και την τυποποίηση του κώδικα. Για το Appify, η επιλογή της Dart είναι στρατηγική: η γλώσσα συνδυάζει **προβλεψιμότητα, ταχύτητα εκτέλεσης και ευκολία ανάλυσης**, καθιστώντας την ιδανική βάση για **παραγωγή κώδικα από πράκτορες τεχνητής νοημοσύνης.**

2.3.4 Dart στην Πρακτική του Appify

Στην πράξη, κάθε agent παράγει τμήματα Dart κώδικα που αντιστοιχούν σε **Widgets, Classes ή State Management Components**. Ο μέσος όρος μήκους ενός αρχείου είναι περίπου **338 tokens**, με τυπική απόκλιση 230, γεγονός που δείχνει ότι ο παραγόμενος κώδικας είναι μικρός, κατανοητός και modular.

2.3.5 Σύγκριση με TypeScript και Kotlin

Η Dart συνδυάζει τη λιτότητα της JavaScript με τη στατική τυποποίηση της TypeScript. Σε σχέση με Kotlin, υπερέχει στην παραγωγή UI λόγω της εγγενούς σύνδεσης με το Flutter SDK. Για το Appify, αυτό σημαίνει **ευκολία αυτόματης δημιουργίας οθονών και μειωμένη πολυπλοκότητα μεταγλώττισης.**

2.4 Το Backend Πλαίσιο FastAPI (Python)

2.4.1 Εισαγωγή

Το FastAPI είναι ένα ελαφρύ αλλά εξαιρετικά αποδοτικό web framework της Python, κατάλληλο για ανάπτυξη REST APIs. Επιλέχθηκε στο Appify λόγω των δυνατοτήτων ασύγχρονης εκτέλεσης (async I/O) και της εγγενούς υποστήριξης Pydantic για τύπους δεδομένων.

2.4.2 Αρχιτεκτονική FastAPI

Το FastAPI βασίζεται στο μοντέλο ASGI και επιτρέπει την παράλληλη εξυπηρέτηση πολλαπλών αιτημάτων. Οι πράκτορες του Appify επικοινωνούν με το backend στέλνοντας JSON δεδομένα σε συγκεκριμένα endpoints. [38]

2.4.3 Ενδεικτικά Endpoints του Appify

- GET /welcome_message: δημιουργεί σύντομο μήνυμα υποδοχής μέσω AI agent.
- POST /validate_idea: λαμβάνει περιγραφή ιδέας και ενεργοποιεί το StartupFlow.

- POST /multimodal_prompt: δέχεται εικόνα και κείμενο, εκτελεί multimodal LLM inference.
- POST /process_feedback: χρησιμοποιείται για βελτιώσεις ή διορθώσεις εφαρμογών.

2.4.4 Διαχείριση Δεδομένων και Pydantic Models

Τα αιτήματα και οι αποκρίσεις μοντελοποιούνται με Pydantic classes (π.χ. Idea, FeedbackRequest). Αυτή η προσέγγιση επιτρέπει έλεγχο εγκυρότητας δεδομένων και αυτόματη τεκμηρίωση μέσω OpenAPI schema. [39]

2.4.5 Επιδόσεις και Επεκτασιμότητα

Χάρη στο Uvicorn server, το backend υποστηρίζει χιλιάδες αιτήματα ανά δευτερόλεπτο με ελάχιστο latency. Η ασύγχρονη φύση του επιτρέπει την ταυτόχρονη εκτέλεση πολλών Flows (agents) χωρίς μπλοκάρισμα. [40]

2.4.6 Ενσωμάτωση με τους Agents

Η FastAPI συνδέεται με το σύστημα πρακτόρων μέσω εσωτερικών Python modules (lib/backend/agents). Όταν ενεργοποιείται ένα endpoint, καλούνται συναρτήσεις όπως StartupFlow.run(), οι οποίες με τη σειρά τους συντονίζουν τα tasks των agents.

2.5 Η πλατφόρμα Firebase

2.5.1 Γενικά

Η Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών που παρέχεται από την Google και προσφέρει ένα πλήρες σύνολο εργαλείων για την κατασκευή, τη βελτίωση και τη διαχείριση εφαρμογών ιστού και κινητών συσκευών. Η Firebase παρέχει μια σειρά από υπηρεσίες όπως βάσεις δεδομένων σε πραγματικό χρόνο, αποθήκευση αρχείων, ανάλυση χρηστών και διαχείριση ειδοποιήσεων. Ένα από τα πιο σημαντικά εργαλεία που προσφέρει είναι το Firebase Authentication [28].

2.5.2 Firebase Authentication

Η Firebase Authentication είναι μια υπηρεσία που επιτρέπει στους προγραμματιστές να ενσωματώσουν μηχανισμούς αυθεντικοποίησης (authentication) σε εφαρμογές ιστού και κινητών συσκευών. Η αυθεντικοποίηση είναι μια κρίσιμη διαδικασία που επιτρέπει στους χρήστες να δημιουργούν λογαριασμούς, να συνδέονται στην εφαρμογή και να έχουν πρόσβαση σε εξατομικευμένες λειτουργίες και δεδομένα. Ένα από τα βασικότερα χαρακτηριστικά της υπηρεσίας είναι τα έτοιμα API και SDK που παρέχει, τα οποία εξασφαλίζουν την εύκολη και γρήγορη ενσωμάτωση της υπηρεσίας σε εφαρμογές Android, iOS και Web, επιτρέποντας την εύκολη διαχείριση χρηστών

και την αυθεντικοποίηση. Ένα ακόμη πολύ σημαντικό της χαρακτηριστικό είναι η υποστήριξη ποικίλων τρόπων αυθεντικοποίησης. Συγκεκριμένα, υποστηρίζονται οι εξής τρόποι:

- Ηλεκτρονικό ταχυδρομείο και Κωδικός (Email and Password): Η πιο συνηθισμένη μέθοδος αυθεντικοποίησης που επιτρέπει στους χρήστες να δημιουργούν λογαριασμούς χρησιμοποιώντας τη διεύθυνση email τους και έναν κωδικό πρόσβασης.
- OAuth πάροχοι: Υποστηρίζει σύνδεση μέσω τρίτων παρόχων όπως Google, Facebook, Twitter, GitHub.
- Αυθεντικοποίηση μέσω Τηλεφώνου (Phone Authentication): Υποστήριξη αυθεντικοποίησης μέσω αριθμού τηλεφώνου με χρήση SMS για επαλήθευση.
- Ανώνυμη Αυθεντικοποίηση (Anonymous Authentication): Παρέχει τη δυνατότητα στους χρήστες να χρησιμοποιούν προσωρινά την εφαρμογή χωρίς να συνδεθούν, επιτρέποντας την εύκολη μετάβαση σε πλήρη αυθεντικοποίηση αργότερα.
- Πάροχος Προσαρμοσμένων Διακριτικών (Custom Token Provider): Ένας τέτοιος πάροχος επιτρέπει στον προγραμματιστή να χρησιμοποιήσει το δικό του μηχανισμό αυθεντικοποίησης και να δημιουργήσει προσαρμοσμένα (custom) JWT tokens, τα οποία στη συνέχεια η Firebase Authentication μπορεί να επαληθεύσει (verify). Με άλλα λόγια παρέχεται στον προγραμματιστή η δυνατότητα χρήσης του δικού του backend ως σύστημα ελέγχου ταυτότητας (για παράδειγμα, μια δική του βάση δεδομένων χρηστών) και στη συνέχεια να δημιουργήσει ένα custom token, το οποίο η Firebase θα χρησιμοποιήσει για να επαληθεύσει την ταυτότητα του χρήστη. Αυτός είναι ο τρόπος που αξιοποιείται στα πλαίσια του παρόντος, όπως θα αναλυθεί στο Κεφάλαιο 3.

Τέλος, αξίζει να τονιστεί ότι η Firebase Authentication προσφέρει μια ασφαλή πλατφόρμα αυθεντικοποίησης που συμμορφώνεται με βέλτιστες πρακτικές ασφαλείας, συμπεριλαμβανομένης της κρυπτογράφησης κωδικών πρόσβασης και της επαλήθευσης ταυτότητας χρηστών.

2.6 Η Πλατφόρμα Docker

2.6.1 Γενικά για το Docker

Το **Docker** επιτρέπει την απομόνωση εφαρμογών σε ελαφριά containers που περιλαμβάνουν όλα τα απαραίτητα εργαλεία και βιβλιοθήκες. Για το Appify, αυτή η τεχνολογία εξασφαλίζει ότι το περιβάλλον ανάπτυξης είναι **αναπαραγώγιμο** και **πλήρως ελεγχόμενο**.

2.6.2 Dockerfile και Docker Compose του Appify

Το Dockerfile της πλατφόρμας βασίζεται στο image mobiledevops/flutter-sdk-image:latest, το οποίο περιέχει το Flutter SDK και όλα τα toolchains και μπορεί να κάνει build εφαρμογές σε linux, mobile, web. [41]

Το docker-compose.yml ορίζει το service flutter-dev, με παραμέτρους: [42]

- ports: 8080 (web app), 9100 (debug),
- volumes: αντιστοίχιση του φακέλου xapp/ για άμεση πρόσβαση των agents,
- devices: /dev/kvm για hardware acceleration,
- environment variables: DISPLAY, UID, GID.

2.6.3 Tmux Session και Hot Reload

Μέσα στο container εκκινείται αυτόματα ένα **tmux session** (ονομαζόμενο mysession), το οποίο διατηρεί ενεργή την εκτέλεση του Flutter web server. Οι πράκτορες μπορούν να στείλουν εντολές σε αυτό το session (π.χ. r για reload) χρησιμοποιώντας Python Docker SDK και tmux automation. Έτσι εξασφαλίζεται η **συνεχής ανανέωση του κώδικα** χωρίς διακοπή υπηρεσίας. [43]

2.6.4 Δικτυακή Δομή και Debugging

Το περιβάλλον επιτρέπει πρόσβαση στο web app μέσω θύρας 8080 και στο Dart Observatory μέσω 9100. Το backend FastAPI ακούει στη θύρα 5000.
Η επικοινωνία container ↔ host διευκολύνεται από την παράμετρο extra_hosts: host.docker.internal:host-gateway.

2.6.5 Πλεονεκτήματα και Περιορισμοί

Πλεονεκτήματα:

- Πλήρης φορητότητα και αναπαραγωγιμότητα περιβάλλοντος.
- Εύκολη ενσωμάτωση εργαλείων CI/CD.
- Ασφαλής απομόνωση διεργασιών.

Περιορισμοί:

- Η εκτέλεση σε privileged mode ενδέχεται να εισάγει κινδύνους ασφάλειας.
- Οι πόροι CPU/GPU του host ενδέχεται να περιορίσουν την ταυτόχρονη εκτέλεση πολλών πρακτόρων.

2.7 Η γλώσσα προγραμματισμού Python

Η **Python** είναι μια δυναμική, υψηλού επιπέδου και γενικού σκοπού γλώσσα προγραμματισμού, η οποία έχει καθιερωθεί ως ένα από τα πιο ευέλικτα και ισχυρά εργαλεία στον χώρο της μηχανικής μάθησης, της τεχνητής νοημοσύνης, της ανάλυσης δεδομένων και της ανάπτυξης εφαρμογών. Η δημοτικότητά της οφείλεται στη **σαφή σύνταξή** της, την **ευκολία μάθησης** και το **εκτεταμένο οικοσύστημα** βιβλιοθηκών που υποστηρίζει σχεδόν κάθε πεδίο εφαρμογής της επιστήμης των υπολογιστών.

Η Python διαθέτει πολυπαραδειγματικό χαρακτήρα, καθώς υποστηρίζει αντικειμενοστραφή, διαδικαστικό και συναρτησιακό προγραμματισμό, ενώ παραμένει συμβατή με άλλες γλώσσες και περιβάλλοντα, μέσω bindings όπως C/C++, Java και Rust. Χάρη στη δυνατότητά της να λειτουργεί σε οποιαδήποτε πλατφόρμα (Windows, Linux, macOS) και να ενοποιεί διαφορετικά συστήματα μέσω απλών διεπαφών (π.χ. REST APIs ή sockets), επιλέχθηκε ως η κύρια γλώσσα υλοποίησης του backend και των πρακτόρων του συστήματος **Appify**.

Στο πλαίσιο της συγκεκριμένης διπλωματικής, η Python χρησιμοποιήθηκε για:

- την ανάπτυξη του **πυρήνα των AI Agents**,
- την ενσωμάτωση των **LLM μοντέλων** μέσω APIs,
- τη διαχείριση Docker containers και
- την αυτοματοποίηση διαδικασιών ανάπτυξης, ανάλυσης και build εφαρμογών.

Η απλότητα της γλώσσας επέτρεψε την ταχεία υλοποίηση σύνθετων διεργασιών, όπως η αυτόματη παραγωγή κώδικα σε πραγματικό χρόνο και η ενορχήστρωση πολλαπλών agents σε παράλληλες ροές (flows).

2.7.1 Το Οικοσύστημα της Python για Τεχνητή Νοημοσύνη

Η Python είναι η **κυρίαρχη γλώσσα στον χώρο της Τεχνητής Νοημοσύνης (AI),** καθώς προσφέρει βιβλιοθήκες και εργαλεία για επεξεργασία φυσικής γλώσσας, ανάλυση δεδομένων, μηχανική μάθηση, deep learning, και αυτοματοποίηση.

Στο Appify αξιοποιήθηκαν αρκετές βιβλιοθήκες της Python, οι οποίες εξειδικεύονται στην αλληλεπίδραση με μεγάλα γλωσσικά μοντέλα (LLMs), στον συντονισμό πρακτόρων και στη διαχείριση περιβάλλοντος ανάπτυξης: [44] [45] [46] [47] [48] [49] [2]

- **CrewAI:** για τη δημιουργία και τον συντονισμό πρακτόρων (agents).
- **LiteLLM:** για την ενοποιημένη διασύνδεση με διάφορα LLMs (OpenAI, Vertex AI, Anthropic).
- **Serper.dev:** για web-based αναζητήσεις (web search agents).
- **GitPython:** για την αυτοματοποίηση της διαχείρισης κώδικα και εκδόσεων.
- **Docker SDK for Python:** για τη διαχείριση container και tmux sessions.
- **Pandas / NumPy:** για την ανάλυση στατιστικών δεδομένων και την αξιολόγηση αποτελεσμάτων (π.χ. μέση τιμή tokens ανά αρχείο Dart).

Αυτή η συλλογή εργαλείων επιτρέπει στην Python να λειτουργεί ως “ενδιάμεσος εγκέφαλος” μεταξύ του περιβάλλοντος τεχνητής νοημοσύνης και των εργαλείων ανάπτυξης λογισμικού.

2.7.2 Η βιβλιοθήκη CrewAI

Η **CrewAI** είναι μια βιβλιοθήκη Python ανοιχτού κώδικα που παρέχει ένα δομημένο πλαίσιο για τη δημιουργία και συνεργασία **πολλαπλών πρακτόρων τεχνητής νοημοσύνης (multi-agent systems).** Αναπτύχθηκε για να διευκολύνει την οργάνωση πολύπλοκων εργασιών σε υποκαθήκοντα, στα οποία κάθε πράκτορας έχει διακριτό ρόλο και λειτουργεί με συγκεκριμένα εργαλεία, μνήμη και στόχο.

Στο Appify, η CrewAI χρησιμοποιείται για να συντονίζει το σύνολο των ενεργειών που απαιτούνται ώστε μια ιδέα να μετατραπεί σε λειτουργική εφαρμογή. Η λογική της βιβλιοθήκης βασίζεται στη δημιουργία:

- **Agents (Πρακτόρων):** μονάδες λογισμικού που διαθέτουν ρόλους, στόχους, και πρόσβαση σε εργαλεία (tools).
- **Tasks:** επιμέρους καθήκοντα που ανατίθενται σε **agents** και εκτελούνται σειριακά ή παράλληλα.
- **Flows:** ροές που περιγράφουν την αλληλουχία των tasks και τον τρόπο που τα αποτελέσματα κάθε βήματος επηρεάζουν τα επόμενα.

Η CrewAI υποστηρίζει τη χρήση διαφορετικών LLMs για κάθε πράκτορα, επιτρέποντας εξατομικευμένες συμπεριφορές και ρυθμίσεις.
Η επικοινωνία μεταξύ των πρακτόρων και των εργαλείων βασίζεται στη μέθοδο **REACT (Reason + Act),** η οποία αποτελεί θεμέλιο για την ικανότητα των πρακτόρων να εκτελούν λογικές και φυσικές ενέργειες.

2.7.3 Το Μοντέλο REACT και η Δομή Thought-Action-Observation

Η μέθοδος **REACT (Reason + Act**) χρησιμοποιείται για τη βελτίωση της απόδοσης των πρακτόρων σε περιβάλλοντα όπου χρειάζεται λογική συλλογιστική σε συνδυασμό με δράση. Στην CrewAI, και κατ' επέκταση στο Appify, κάθε agent λειτουργεί σε κύκλο αλληλεπίδρασης που αποτελείται από τα εξής στάδια:

1. **Thought (Σκέψη):**
   Ο πράκτορας αναλύει το πρόβλημα ή το ερώτημα που έχει τεθεί. Εδώ παράγει εσωτερικές λογικές προτάσεις, υποθέσεις και επεξηγήσεις για το πώς θα προχωρήσει.
2. **Action (Ενέργεια):**
   Με βάση τη σκέψη, ο πράκτορας επιλέγει μια ενέργεια, η οποία συνήθως αντιστοιχεί στη χρήση ενός εργαλείου. Για παράδειγμα, "Action: RelativeFileWriterTool" ή "Action: WebSearch".
3. **Action Inputs (Είσοδοι Ενέργειας):**
   Τα δεδομένα που χρειάζεται για να εκτελέσει την ενέργεια (π.χ. περιεχόμενο αρχείου, όνομα path, όρους αναζήτησης).
4. **Observation (Παρατήρηση):**
   Ο πράκτορας λαμβάνει το αποτέλεσμα της ενέργειας του εργαλείου (π.χ. "το αρχείο γράφτηκε επιτυχώς", "η αναζήτηση επέστρεψε 5 αποτελέσματα").
5. **Thought → Answer (Τελική Απόκριση):**
   Ο πράκτορας συνθέτει την τελική του απάντηση ή συμπέρασμα, με βάση τις παρατηρήσεις του και τη γενική στρατηγική επίλυσης.

Ένα τυπικό παράδειγμα ροής REACT όπως καταγράφεται στα logs του Appify έχει τη μορφή:

```
Thought: To implement the login feature, I should create a new
screen.
Action: RelativeFileWriterTool
Action Input: "lib/screens/login_screen.dart"
Observation: File created successfully.
Thought: Now I should update the navigation route.
Action: RelativeFileWriterTool
Action Input: "lib/main.dart"
Observation: Route updated successfully.
Answer: Login screen implemented and integrated.
```

Η παραπάνω δομή επιτρέπει αυτορρύθμιση και αυτοδιόρθωση: κάθε agent αντιλαμβάνεται τα λάθη του και επαναλαμβάνει ενέργειες έως ότου επιτύχει το στόχο του. Αυτό προσδίδει στην πλατφόρμα ιδιότητες "ενεργού προγραμματιστή" που μαθαίνει από το περιβάλλον εκτέλεσης.

Οι πρόσφατες εργασίες του Τζαχρήστα και συνεργατών ενισχύουν το επιχείρημα ότι οι LLM-based agents αποτελούν μια πρακτική και επεκτάσιμη αρχιτεκτονική για πραγματικές εφαρμογές. Συγκεκριμένα, προτείνεται μια υψηλού επιπέδου μεθοδολογία κατασκευής AI-agents βάσει LLM με αξιοποίηση APIs και ρητά βήματα ανάλυσης εργασιών και σημασιολογικής ευθυγράμμισης (Τζαχρήστας, 2024). Παράλληλα, στο πλαίσιο της ασφάλειας συστημάτων, παρουσιάζεται μια πολυ-LLM ετερογενής αρχιτεκτονική (“Perses”) που δείχνει πώς μικρότερα μοντέλα μπορούν να συνεργαστούν για να επιτύχουν δισείσδυση σε ένα λειτουργικό σύστημα (Weber, Tzachristas & Sui, 2025). Τέλος, σε εφαρμοσμένο κοινωνικοτεχνικό πρόβλημα, εισάγεται καθοδηγούμενη, μεθοδολογία βασισμένη σε ”personas” για παραγωγή τεχνητών ερευνών μεγάλης κλίμακας, καταδεικνύοντας τη χρησιμότητα agentic ροών σε πεδία πέραν της καθαυτό πληροφορικής [50]

2.7.4 Ο Developer Agent και τα Εργαλεία του

Ο **Developer Agent** είναι ο κεντρικός πράκτορας υπεύθυνος για την παραγωγή και διαχείριση του κώδικα της εφαρμογής.
Διαθέτει πρόσβαση σε πληθώρα εργαλείων (tools) τα οποία του επιτρέπουν να διαβάζει, γράφει, αναλύει, και διορθώνει αρχεία κώδικα μέσα στο Dockerized περιβάλλον.

Κύρια εργαλεία:

- **RelativeFileWriterTool:** δημιουργεί ή τροποποιεί αρχεία Dart.
- **RelativeFileReadTool:** διαβάζει υπάρχοντα αρχεία για ανάλυση και ενημέρωση.
- **FlutterPubAddPackagesTool:** εγκαθιστά πακέτα μέσω flutter pub add.
- **FlutterAnalyzeTool:** εκτελεί στατική ανάλυση κώδικα και build validation.
- **Git Commit Tools:** δημιουργεί αυτόματα commits, χρησιμοποιώντας περιγραφές που παράγει ένα μικρότερο LLM (LiteLLM).

Ο Developer Agent λειτουργεί ως συνδυασμός **REACT agent** και **Tool-augmented LLM**, καθώς συνδυάζει την ικανότητα λογικής επεξεργασίας με απτές ενέργειες στο filesystem.

2.7.5 Η Βιβλιοθήκη LiteLLM

Η **LiteLLM** αποτελεί ένα ενιαίο API που επιτρέπει τη χρήση πολλαπλών LLM παρόχων (OpenAI, Vertex AI, Anthropic) με κοινό interface.
Στο Appify χρησιμοποιείται:

- για την παραγωγή μικρών, lightweight αποκρίσεων (π.χ. σύντομα commit messages),
- για βοηθητικούς agents (όπως ο Welcome Agent ή ο Comment Generator).

Με τη LiteLLM εξασφαλίζεται **ομοιομορφία στη χρήση μοντέλων** και **ανεξαρτησία από πάροχο**, γεγονός που διευκολύνει τη μελλοντική προσαρμογή της πλατφόρμας.

Μειονέκτημα του είναι ότι για να έχει αυτή την ομοιομορφία πολλές φορές μένει μερικές μέρες ή βδομάδες πίσω στην προσθήκη των νέων εργαλείων από κάθε πάροχο.

2.7.6 Εργαλεία Αναζήτησης και Εμπλουτισμού Πληροφορίας

Η βιβλιοθήκη **Serper.dev** προσφέρει τη δυνατότητα σε agents να πραγματοποιούν αναζητήσεις στο διαδίκτυο, να αντλούν αποτελέσματα και να τα ενσωματώνουν στις απαντήσεις τους.

Στο Appify, η χρήση της βιβλιοθήκης αυτής επιτρέπει:

- **Context-aware ενημέρωση** για frameworks, βιβλιοθήκες ή API συντάξεις.
- **Ανάκτηση παραδειγμάτων** ή τεκμηρίωσης που βοηθούν στην παραγωγή ακριβέστερου κώδικα.

Ο μηχανισμός ενσωματώνει τεχνικές **web scraping** και **contextual fusion**, επιτρέποντας στο LLM να έχει “ενεργή πρόσβαση” σε ενημερωμένες πληροφορίες, υπερβαίνοντας τους περιορισμούς της offline γνώσης.

2.7.7 Παρακολούθηση και Τηλεμετρία

Για τη βελτίωση της αξιοπιστίας και της διαφάνειας του συστήματος, η πλατφόρμα χρησιμοποιεί το **AgentOps**, ένα εργαλείο τηλεμετρίας που συλλέγει μετρικές όπως χρόνος εκτέλεσης, αριθμός ενεργειών ανά flow, επιτυχία builds και ποσοστά επαναλήψεων.
Τα δεδομένα αυτά αποθηκεύονται για μελλοντική **αξιολόγηση απόδοσης** και **βελτιστοποίηση των πρακτόρων**.

Κεφάλαιο 3: Δομή και μεθοδολογία υλοποίησης του συστήματος

3.1 Εισαγωγή

Το παρόν κεφάλαιο περιγράφει αναλυτικά τη μεθοδολογία που ακολουθήθηκε για την υλοποίηση της πλατφόρμας Appify, καθώς και τη δομή του συνολικού συστήματος. Στόχος είναι η παρουσίαση της διαδικασίας ανάπτυξης βήμα προς βήμα, από τον αρχικό σχεδιασμό έως τη δημιουργία των πρώτων λειτουργικών εφαρμογών που παρήχθησαν μέσω των πρακτόρων τεχνητής νοημοσύνης.

Η ανάπτυξη του Appify πραγματοποιήθηκε μέσα από μια σειρά επαναληπτικών κύκλων (iterations), όπου κάθε κύκλος παρήγαγε μετρήσιμα αποτελέσματα και συνέβαλε στην ωρίμανση της αρχιτεκτονικής. Κατά τη διάρκεια της υλοποίησης, δόθηκε ιδιαίτερη έμφαση στην ευελιξία, στην προσαρμοστικότητα του συστήματος σε διαφορετικές περιπτώσεις χρήσης, καθώς και στην αντικειμενική αξιολόγηση της απόδοσης των πρακτόρων.

Στα επόμενα τμήματα παρουσιάζονται αναλυτικά η μεθοδολογία ανάπτυξης, ο τρόπος συλλογής και αξιοποίησης των δεδομένων, η συνολική αρχιτεκτονική του Appify και η ανάλυση των επιμέρους υποσυστημάτων που το συνθέτουν.

3.2 Μεθοδολογία Ανάπτυξης (Agile / Iterative Approach)

Για την ανάπτυξη της πλατφόρμας Appify υιοθετήθηκε μια ευέλικτη (Agile) προσέγγιση ανάπτυξης λογισμικού, με κύρια χαρακτηριστικά τη σταδιακή εξέλιξη του έργου, την προσαρμοστικότητα και τη συνεχή αξιολόγηση. Η επιλογή αυτής της μεθοδολογίας βασίστηκε στην ανάγκη γρήγορης προτυποποίησης (rapid prototyping) και συνεχούς βελτίωσης, καθώς το σύστημα περιλάμβανε πολλαπλά υποσυστήματα (frontend, backend, AI layer, DevOps) που εξελίσσονταν παράλληλα.

Η ανάπτυξη οργανώθηκε σε μικρούς κύκλους εργασίας (sprints) καθένας από τους οποίους είχε σαφώς καθορισμένους στόχους:
στην αρχή η ενοποίηση του περιβάλλοντος ανάπτυξης, έπειτα η σύνδεση των πρακτόρων CrewAI, και στη συνέχεια η δημιουργία και αξιολόγηση των πρώτων αυτοματοποιημένων builds.

Για τη διαχείριση έργου χρησιμοποιήθηκαν δύο βασικά εργαλεία:

- το **Git**, ως σύστημα ελέγχου εκδόσεων (version control), που επέτρεψε την παρακολούθηση αλλαγών, τη δημιουργία διαφορετικών branches ανά λειτουργία, καθώς και τη συγχώνευση κώδικα μέσω pull requests,
- και το **YouTrack**, ως σύστημα διαχείρισης εργασιών, το οποίο παρείχε **Kanban matrix** για την απεικόνιση της προόδου και των εκκρεμοτήτων.

Με αυτόν τον τρόπο, η ανάπτυξη οργανώθηκε σε **διακριτά καθήκοντα (tasks)** με σαφή κατάσταση (to-do, in progress, done), ενώ ταυτόχρονα το Git παρείχε πλήρες ιστορικό αλλαγών. Ο συνδυασμός αυτών των εργαλείων επέτρεψε τη διατήρηση μιας **δομημένης αλλά ευέλικτης διαδικασίας ανάπτυξης**, που ευθυγραμμίζεται με τις αρχές της μεθοδολογίας Agile.

Κάθε νέο χαρακτηριστικό του συστήματος ενσωματωνόταν με βάση τον κύκλο **Prompt → Ανάπτυξη → Έλεγχος → Διόρθωση → Ενσωμάτωση**. Ο κύκλος αυτός αντιστοιχεί άμεσα στον τρόπο λειτουργίας των ίδιων των πρακτόρων του Appify, οι οποίοι μέσω της μεθόδου **REACT (Reason + Act)** εκτελούν ενέργειες, παρατηρούν αποτελέσματα και προσαρμόζουν τη συμπεριφορά τους. Έτσι, η μεθοδολογία ανάπτυξης και η λειτουργική φιλοσοφία της πλατφόρμας υπήρξαν αλληλένδετες.

Όπως παρουσιάζεται και στην παραπάνω εικόνα, η μεθοδολογία διακρίνεται στα

Εικόνα 6 Μεθοδολογία Agile

εξής στάδια, τα οποία ακολουθούνται επαναληπτικά μέχρι την ολοκλήρωσω του έργου:

- **Πλάνο (Plan):** Το πρώτο βήμα στην ευέλικτη μεθοδολογία είναι η δημιουργία ενός πλάνου, στο οποίο γίνεται ο καθορισμός των βασικών λειτουργιών και απαιτήσεων και ο εκτιμώμενος χρόνος για την σχεδίαση και υλοποίηση αυτών. Αυτές πρέπει να είναι υλοποιήσιμες, ρεαλιστικές, πρωτοποριακές και να εξυπηρετούν τις ανάγκες των χρηστών, αποφεύγοντας περιττά στοιχεία και πλεονασμούς.
- **Σχεδιασμός (Design):** Ακολουθεί ο σχεδιασμός του συστήματος, ο οποίος βασίζεται στις καθορισμένες απαιτήσεις του πρώτου σταδίου και στη διαθεσιμότητα πόρων, όπως χρήματα, εξοπλισμός, εργαλεία και χρόνος. Κάθε νέα απαίτηση επηρεάζει τον σχεδιασμό, οδηγώντας στην ανάγκη εύρεσης της κατάλληλης δομής για την υλοποίηση της.
- **Ανάπτυξη (Development):** Το τρίτο στάδιο αποτελεί η ανάπτυξη, η οποία έγινε σταδιακά, με την ολοκλήρωση κάθε φορά συγκεκριμένου τμήματος, ώστε αυτό να ενσωματώνεται ομαλά στο σύνολο, εξασφαλίζοντας ομοιογένεια. Παράλληλα γινόντουσαν και οι τυχόν απαραίτητες αλλαγές στα ήδη υπάρχοντα τμήματα. Το GitHub χρησιμοποιήθηκε για την ασφαλή αποθήκευση του κώδικα, την παρακολούθηση της εξέλιξης και τη διατήρηση ιστορικού και παλαιότερων εκδόσεων.

- **Δοκιμή και Διάθεση (Testing and Deployment)** Μετά την ανάπτυξη κάθε λειτουργίας, ακολουθεί δοκιμή αρχικά σε τοπικό περιβάλλον και στη συνέχεια υπό πραγματικές συνθήκες, για να διασφαλιστεί η ορθότητα και η αποδοτικότητα της εφαρμογής. Το deployment γινόταν μόνο μετά από επιτυχημένο testing.
- **Αξιολόγηση (Review):** Το τελικό στάδιο περιλαμβάνει την αξιολόγηση των αποτελεσμάτων και του τρόπου ανάπτυξης, με στόχο τον εντοπισμό προβλημάτων και την αποφυγή τους σε μελλοντικές λειτουργίες. Πολλοί επανέλεγχοι πραγματοποιήθηκαν για να διασφαλιστεί η ακρίβεια και η ορθότητα του τελικού συστήματος.

3.3 Μέθοδος συλλογής δεδομένων

3.3.1 Προέλευση των Prompts

Για την αξιολόγηση και τη σταδιακή εκπαίδευση του συστήματος, χρησιμοποιήθηκαν πραγματικά αιτήματα χρηστών από το περιβάλλον του ερευνητή. Τα prompts προήλθαν από ανθρώπους που ενδιαφέρονταν να δημιουργήσουν εφαρμογές για προσωπικούς ή επαγγελματικούς σκοπούς.

Συγκεκριμένα, πραγματοποιήθηκαν δύο κύρια πειράματα:

- Το πρώτο αφορούσε τη δημιουργία μιας **e-commerce εφαρμογής** για κεριά σόγιας, η οποία περιλάμβανε παρουσίαση προϊόντων, τιμοκατάλογο και φόρμα επικοινωνίας.
- Το δεύτερο αφορούσε την κατασκευή ενός **ιστοτόπου για παραδοσιακή ταβέρνα**, με προβολή μενού, κρατήσεις και φωτογραφικό περιεχόμενο.

Τα prompts δόθηκαν απευθείας στο σύστημα Appify, το οποίο ανέλαβε τη δημιουργία των αντίστοιχων εφαρμογών μέσω των πρακτόρων του, χωρίς προεπεξεργασία ή τροποποίηση από τον ερευνητή. Με τον τρόπο αυτό εξασφαλίστηκε η αυθεντικότητα και η ανεξαρτησία της αξιολόγησης..

3.3.2 Βελτιστοποίηση των Prompts και Agents

Η διαδικασία ανάπτυξης ακολούθησε μια **test-driven** και **data-driven** προσέγγιση. Στις αρχικές δοκιμές χρησιμοποιήθηκε **ένας ενιαίος πράκτορας**, ο οποίος ήταν υπεύθυνος για όλα τα καθήκοντα — από την κατανόηση της ιδέας μέχρι τη δημιουργία του τελικού κώδικα.

Καθώς η πολυπλοκότητα των εφαρμογών αυξανόταν, η αρχιτεκτονική αυτή αποδείχθηκε περιοριστική. Παρατηρήθηκε ότι οι πράκτορες δυσκολεύονταν να διαχειριστούν ταυτόχρονα πολλά βήματα και να αναγνωρίσουν λάθη σε προηγούμενα στάδια. Έτσι, η μεθοδολογία εξελίχθηκε: τα καθήκοντα **διαχωρίστηκαν σε subtasks** και ανατέθηκαν σε **εξειδικευμένους υποπράκτορες (sub-agents).**

Αυτή η μετατόπιση προς μια **πολυπρακτορική αρχιτεκτονική (multi-agent)** επέτρεψε:
τη διανομή της εργασίας μεταξύ agents με διαφορετικούς ρόλους (IdeaValidator, Architect, Developer, Critic),
την αύξηση της ακρίβειας και της συνέπειας στις απαντήσεις,
και τη σταδιακή βελτίωση των prompts μέσω παρατηρήσεων από τα logs και τα αποτελέσματα.

Ο κύκλος Prompt → Code → Error → Feedback → Improved Prompt αποτέλεσε θεμέλιο της διαδικασίας μάθησης του συστήματος. Ουσιαστικά, η ίδια η πλατφόρμα χρησιμοποιήθηκε για τη βελτίωσή της — μια μορφή μετα-εκπαίδευσης μέσω πράξης (self-improving workflow).

3.3.3 Καταγραφή και Μετρήσεις

Κατά τη διάρκεια των πειραμάτων καταγράφηκαν **μετρικές που σχετίζονται με τη χρήση LLMs** και το κόστος εκτέλεσης.
Ο μέσος αριθμός tokens ανά παραγόμενο αρχείο Dart ήταν **338.5**, με **τυπική απόκλιση 230.2 tokens**, ενώ ανά εφαρμογή η κατανάλωση κυμαινόταν από **3 έως 15 εκατομμύρια tokens.**

Η εκτέλεση πραγματοποιήθηκε με τρία διαφορετικά μοντέλα: **GPT-5, Gemini 2.5 Pro και GPT-4.1-mini**, επιτρέποντας τη σύγκριση κόστους και ταχύτητας απόκρισης. Παρόλο που συλλέχθηκαν logs από τις διαδικασίες build και test, δεν πραγματοποιήθηκε ποσοτική ανάλυση των αποτελεσμάτων, καθώς ο στόχος ήταν η ποιοτική αξιολόγηση της συμπεριφοράς των πρακτόρων.

Τα δεδομένα αυτά προσφέρουν μια πρώτη εκτίμηση για τη **λογική κλίμακα** κόστους και την **υπολογιστική επιβάρυνση** της πλατφόρμας, συμβάλλοντας στον μελλοντικό σχεδιασμό πιο αποδοτικών agents.

3.4 Η Αρχιτεκτονική του συστήματος

Η συνολική αρχιτεκτονική του Appify ακολουθεί **πολυεπίπεδη λογική**, διαχωρίζοντας σαφώς τις ευθύνες μεταξύ frontend, backend και επιπέδου τεχνητής νοημοσύνης.

Στο ανώτερο επίπεδο βρίσκεται το **Frontend**, αναπτυγμένο σε **Flutter/Dart**, το οποίο λειτουργεί ως διεπαφή χρήστη και παράγει τα τελικά προϊόντα (εφαρμογές). Το frontend επικοινωνεί μέσω **RESTful API** με το **Backend**, το οποίο έχει υλοποιηθεί σε **Python/FastAPI**.

Το backend φιλοξενεί το **AI Engine**, που βασίζεται στη βιβλιοθήκη **CrewAI**. Εκεί υλοποιείται η λογική των πρακτόρων, η κατανομή των εργασιών, και η διαχείριση των αποτελεσμάτων. Οι πράκτορες επικοινωνούν με εξωτερικά APIs, όπως **OpenAI, Vertex AI, Serper.dev** και **AgentOps**, για πρόσβαση σε μοντέλα, αναζητήσεις και τηλεμετρία.

Η διαδικασία build και δοκιμής πραγματοποιείται εντός **Docker containers**, όπου κάθε Flutter project εκτελείται σε απομονωμένο περιβάλλον με χρήση **tmux sessions**. Η επικοινωνία μεταξύ των επιπέδων επιτυγχάνεται μέσω ασφαλούς ανταλλαγής δεδομένων (volumes και sockets).

(Εικόνα 3.1 – Συνολική αρχιτεκτονική του συστήματος Appify)

Η αρχιτεκτονική αυτή εξασφαλίζει **φορητότητα, επαναληψιμότητα** και **ευκολία επεκτασιμότητας**, επιτρέποντας την ενσωμάτωση νέων πρακτόρων ή εργαλείων χωρίς τροποποίηση του πυρήνα.

3.5 Ανάλυση των Συστατικών Μερών του Συστήματος

3.5.1 Σύστημα Πρακτόρων (CrewAI Framework)

Η βιβλιοθήκη **CrewAI** αποτέλεσε το θεμέλιο του συστήματος πρακτόρων. Κάθε πράκτορας ορίζεται μέσω decorators (@agent, @task, @crew) και διαθέτει δική του κατάσταση, ρόλο και σύνολο εργαλείων.

Η συνεργασία των πρακτόρων υλοποιείται μέσω της μεθόδου **REACT (Reason + Act)**, όπου κάθε agent ακολουθεί τον κύκλο Thought → Action → Observation → Answer. Με τον τρόπο αυτό, το Appify προσομοιώνει τη διαδικασία ανθρώπινης λογικής: κάθε πράκτορας αναλογίζεται τη δράση του, ενεργεί, παρατηρεί το αποτέλεσμα και προσαρμόζει τη στρατηγική του.

Η μετάβαση από μονοπρακτορική σε πολυπρακτορική αρχιτεκτονική αποτέλεσε κομβικό σημείο της υλοποίησης, καθώς αύξησε τη σταθερότητα και την ακρίβεια των αποτελεσμάτων.

3.5.2 Περιβάλλον Docker και Αυτοματισμοί

Η χρήση Docker εξασφάλισε πλήρη απομόνωση και αναπαραγωγιμότητα του περιβάλλοντος ανάπτυξης. Κάθε build εκτελούνταν μέσα σε container βασισμένο στην εικόνα mobiledevops/flutter-sdk-image:latest.

Μέσα στο container εκκινείτο αυτόματα tmux session, το οποίο κρατούσε ενεργό τον Flutter web server. Οι πράκτορες μπορούσαν να στείλουν εντολές στο session (π.χ. r για reload) μέσω **Docker SDK for Python**, επιτυγχάνοντας **συνεχή ανάπτυξη σε πραγματικό χρόνο (hot reload).**

Η επικοινωνία μεταξύ containers και host επιτρέπει την απρόσκοπτη λειτουργία μεταξύ FastAPI, Flutter και CrewAI modules.

3.5.3 Εφαρμογή Flutter – Προεπισκόπηση και Παραγόμενος Κώδικας

Κάθε εφαρμογή που παράγεται από το Appify περιλαμβάνει πλήρη δομή Flutter project: φακέλους lib/, assets/, test/ και βασικό αρχείο main.dart. Οι πράκτορες δημιουργούν αυτόματα νέα widgets, routes και screens, ενώ ο χρήστης μπορεί να δει την πρόοδο μέσω browser με **άμεση ανανέωση περιεχομένου**.

Η παραγωγή του κώδικα είναι **καθαρή και επεκτάσιμη,** με σαφή ονομασία αρχείων και αυτόματη ενσωμάτωση πακέτων μέσω flutter pub add.

3.5.4 Backend FastAPI Server

Το backend διαχειρίζεται τα αιτήματα του χρήστη και ενεργοποιεί τις αντίστοιχες ροές πρακτόρων.
Κύρια endpoints:

/validate_idea για έλεγχο και αρχικοποίηση project,

/process_feedback για βελτιώσεις και επαναλήψεις.

Τα δεδομένα ανταλλάσσονται σε μορφή JSON και επιβεβαιώνονται μέσω **Pydantic models.**

3.5.5 DevOps – Git, Build και Παρακολούθηση

Η ανάπτυξη συνοδεύτηκε από ολοκληρωμένο **Git workflow**, με branching ανά agent και αυτόματη δημιουργία **commit messages** μέσω **LiteLLM**.

Οι διαδικασίες build ελέγχονταν με το εργαλείο **FlutterAnalyzeTool**, το οποίο εξασφάλιζε την ορθότητα του κώδικα πριν τη δημιουργία κάθε εφαρμογής. Τα αποτελέσματα των builds αποθηκεύονταν σε logs, ενώ η συνολική τηλεμετρία παρακολουθούταν από το **AgentOps**.

3.6 Κόστος Υλοποίησης της Διπλωματικής Εργασίας

3.6.1 Περιβάλλοντα ανάπτυξης

Όλες οι δοκιμές πραγματοποιήθηκαν σε **φορητό υπολογιστή** με λειτουργικό **Windows** και **WSL**, χρησιμοποιώντας **Docker Desktop, Android Studio** και **PyCharm Professional.**
Η πλατφόρμα συνδέθηκε με εξωτερικά APIs των **OpenAI**, **Vertex AI**, **AgentOps**, **Serper.dev** και **Maxim Monitoring**, καλύπτοντας όλο το φάσμα από τη δημιουργία έως την παρακολούθηση εφαρμογών.

3.6.2 Υλοποίηση Frontend

Η ανάπτυξη του frontend βασίστηκε στο **Flutter SDK** και στο **Dart environment**, με πλήρη υποστήριξη hot reload και real-time preview. Το build pipeline διαχειριζόταν από Docker και ολοκληρωνόταν χωρίς χειροκίνητες επεμβάσεις.

3.6.4 Υλοποίηση του Backend και Πρακτόρων

Η υλοποίηση των πρακτόρων πραγματοποιήθηκε σε Python, συνδυάζοντας **FastAPI** με το πλαίσιο **CrewAI**, ενώ ο έλεγχος των containers και η εκτέλεση εντολών εντός του απομονωμένου περιβάλλοντος έγινε μέσω **Docker SDK**. Στα πειράματα, κάθε εφαρμογή κατανάλωνε κατά προσέγγιση **3 έως 15 εκατομμύρια tokens**, με αναλογία περίπου **80% input και 20% output**. Κατά τη διάρκεια των δοκιμών πραγματοποιήθηκαν συνολικά **περίπου 200-300 εκτελέσεις**, χρησιμοποιώντας διαφορετικά μοντέλα (GPT-5, Gemini 2.5 Pro, GPT-4.1-mini) και διαφοροποιώντας το εύρος των tokens ανάλογα με την πολυπλοκότητα του έργου.

Με βάση τις τρέχουσες τιμές ανά εκατομμύριο tokens - για παράδειγμα, **GPT-5: 1.25 USD / 1M input και 10 USD / 1M output** (σύμφωνα με την επίσημη τιμολόγηση της OpenAI) - το κόστος **ανά εφαρμογή κυμαινόταν περίπου 9 έως 45 δολάρια.** Λαμβάνοντας υπόψη ότι τα πειράματα κάλυψαν ένα μικτό σύνολο μοντέλων και μεγεθών εφαρμογών, το συνολικό κόστος εκτέλεσης εκτιμάται ότι κυμάνθηκε μεταξύ **1.000 και 1.500 ευρώ**, ποσό που αντανακλά μια ρεαλιστική και βιώσιμη δαπάνη για το ερευνητικό εύρος της παρούσας διπλωματικής εργασίας.

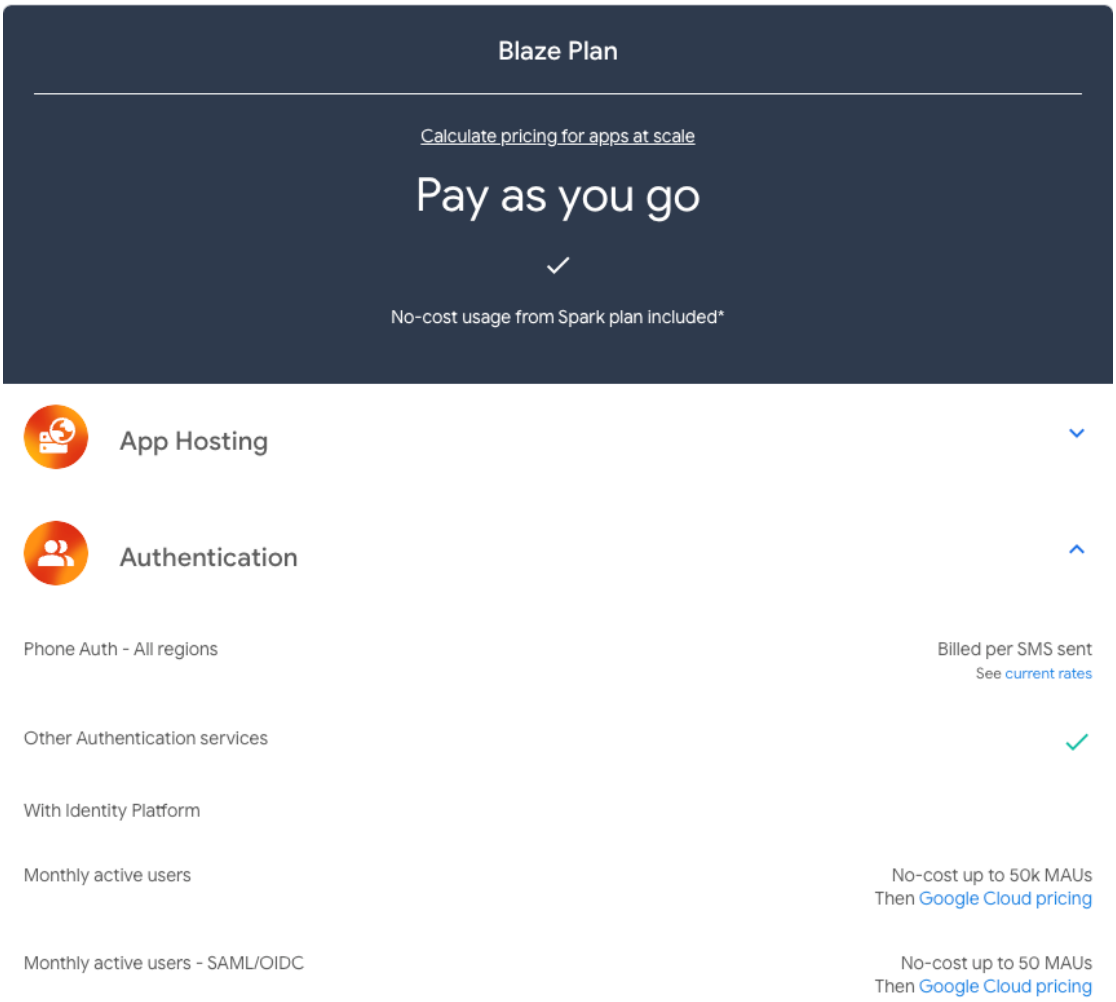
3.6.5 Υπηρεσίες Cloud Firebase (Spark Plan)

Στο τέλος κάθε εφαρμογής πραγματοποιούνταν αυτόματα firebase init και firebase deploy, με αποτέλεσμα η εφαρμογή να φιλοξενείται στο Firebase Hosting.
Το περιβάλλον περιλάμβανε ενεργό SSL, CDN caching και versioned deployments, ενώ η χρήση του Spark Plan επέτρεψε δωρεάν δημοσίευση χωρίς επιπλέον κόστος.

3.6.5 Firebase (Spark Plan)

Στο τέλος κάθε εφαρμογής πραγματοποιούνταν αυτόματα firebase init και firebase deploy, με αποτέλεσμα η εφαρμογή να φιλοξενείται στο Firebase Hosting.
Το περιβάλλον περιλάμβανε ενεργό SSL, CDN caching και versioned deployments, ενώ η χρήση του Spark Plan επέτρεψε δωρεάν δημοσίευση χωρίς επιπλέον κόστος.

Η Firebase παρέχει εντελώς δωρεάν το πλάνο Spark, το οποίο περιλαμβάνει όλες τις βασικές υπηρεσίες που χρειάζονται για την αυθεντικοποίηση των χρηστών στα πλαίσια του παρόντος έργου. Παρακάτω φαίνονται συγκριτικά οι παροχές του δωρεάν πλάνου Spark και του "Pay as you go" (πληρωμή ανά χρήση) πλάνου Blaze σχετικά με την υπηρεσία αυθεντικοποίησης:



Εικόνα 8 Firebase Blaze Plan

3.7 Σύνοψη Κεφαλαίου

Το παρόν κεφάλαιο παρουσίασε αναλυτικά τη μεθοδολογία και τη δομή της υλοποίησης της πλατφόρμας Appify. Η ανάπτυξη ακολούθησε επαναληπτική προσέγγιση, με συνεχή βελτίωση πρακτόρων και σταδιακή ενσωμάτωση νέων εργαλείων.

Η πολυπρακτορική αρχιτεκτονική σε συνδυασμό με το περιβάλλον Docker και τη χρήση LLMs υψηλής απόδοσης επέτρεψε τη δημιουργία ενός συστήματος που μπορεί να παράγει, να ελέγχει και να διαθέτει λειτουργικές εφαρμογές χωρίς ανθρώπινη παρέμβαση στον κώδικα.

Στο επόμενο κεφάλαιο παρουσιάζονται τα αποτελέσματα των πειραμάτων, τα μετρικά αξιολόγησης και η συνολική αποτίμηση της απόδοσης της πλατφόρμας.

Κεφάλαιο 4: Παρουσίαση, ανάλυση και ερμηνεία του κώδικα υλοποίησης

4.1 Εισαγωγή

Το παρόν κεφάλαιο παρουσιάζει, αναλύει και ερμηνεύει την εσωτερική λειτουργία της πλατφόρμας Appify, με έμφαση στη λογική οργάνωση, στις τεχνικές επιλογές και στις ροές εκτέλεσης που καθιστούν εφικτή την αυτόματη παραγωγή εφαρμογών Flutter από περιγραφές σε φυσική γλώσσα. Η Appify υλοποιεί ένα no-code παράδειγμα ευφυούς αυτοματισμού, όπου πράκτορες τεχνητής νοημοσύνης (AI agents) συντονίζονται ώστε να μετασχηματίζουν επιχειρηματικές/λειτουργικές ιδέες σε αναπαράξιμα παραδοτέα λογισμικού. Οι πυρήνες της ευφυΐας υλοποιούνται σε Python, αξιοποιώντας το πλαίσιο CrewAI και το συλλογιστικό πρότυπο REACT (Reason → Act), ενώ η διεπαφή με ετερογενή μεγάλα γλωσσικά μοντέλα (LLMs) γίνεται μέσω της LiteLLM, επιτρέποντας επιλογή μοντέλου ανά είδος εργασίας (π.χ. GPT-5, Gemini 2.5 Pro, GPT-4.1-mini).

Στο backend, η FastAPI λειτουργεί ως λεπτή στιβάδα ενορχήστρωσης (orchestration layer), εκθέτοντας REST endpoints που αντιστοιχούν σε σαφώς ορισμένες ροές πρακτόρων (π.χ. αρχικοποίηση ιδέας, ανάλυση βελτίωσης, παραγωγή πλάνου κώδικα, εφαρμογή αλλαγών). Η εκτέλεση builds και η απομόνωση του περιβάλλοντος ανάπτυξης γίνονται εντός Docker containers, επιτρέποντας αναπαραγωγιμότητα, απομόνωση εξαρτήσεων και ομοιόμορφη συμπεριφορά μεταξύ μηχανών. Το παραγόμενο αποτέλεσμα (κώδικας Dart/Flutter) είναι κατάλληλο για άμεση προεπισκόπηση και διάθεση (π.χ. Firebase Hosting), ώστε ο χρήστης να λαμβάνει γρήγορα ανατροφοδότηση στον κύκλο σχεδίασης.

Ειδικότερα, στο κεφάλαιο αυτό:

- τεκμηριώνεται η πολυπρακτορική ροή (validation → brainstorming → architecting → developing) και ο τρόπος με τον οποίο το REACT επιβάλλει δομή και ιχνηλασιμότητα στη συλλογιστική των πρακτόρων,
- αναλύεται η αρχιτεκτονική κώδικα και η οργάνωση των φακέλων σε modules με σαφείς ευθύνες (flows, crews, tools, helpers, API),
- περιγράφονται οι διεπαφές FastAPI και τα σχήματα δεδομένων εισόδου/εξόδου,
- παρουσιάζεται ο μηχανισμός προεπισκόπησης του παραγόμενου frontend και οι μη-λειτουργικές απαιτήσεις που ικανοποιούνται (παρατηρησιμότητα, επεκτασιμότητα, ανθεκτικότητα, κόστος/απόδοση).

Σκοπός είναι ο αναγνώστης να κατανοήσει πώς τα δομικά μέρη συνεργάζονται σε επίπεδο αρχών, πότε καλούνται, με ποιον τρόπο ανταλλάσσουν τεκμήρια (artifacts) και γιατί οι σχεδιαστικές επιλογές εξυπηρετούν τους στόχους της πλατφόρμας (χαμηλός χρόνος μέχρι πρωτότυπο, ποιότητα, επαναληψιμότητα).

4.2 Κώδικας Backend και Ροή Πρακτόρων (AI Layer)

Η ενότητα εστιάζει στη στοιβάδα ευφυΐας του Appify, όπου το backend επιτελεί ορχήστρωση ροών (flows) και συντονισμό πρακτόρων (crews) με συγκεκριμένες ικανότητες (εργαλεία, πρότυπα tasks). Η FastAPI παρέχει το HTTP interface και τη

συμβασιοποίηση των μηνυμάτων (JSON), ενώ οι αλυσίδες λογικής υλοποιούνται ως συντεθειμένες ροές στο CrewAI, ακολουθώντας τη μεθοδολογία REACT:

- Reason: κάθε agent παράγει ρητή συλλογιστική (thought) σε σχέση με τον στόχο.
- Act: ο agent εκτελεί ελεγχόμενη ενέργεια μέσω εργαλείων (π.χ. ανάγνωση/εγγραφή αρχείων, ανάλυση Flutter, αναζήτηση).
- Observation: τα αποτελέσματα της ενέργειας ενσωματώνονται στο context.
- Answer: συνάγεται συμπέρασμα/έξοδος με καθορισμένη μορφή.

Η ροή Initial App Generation οργανώνεται σε τέσσερα στάδια:

1. Idea Validation (σκοπιμότητα, ασφάλεια, αξία) με ελεγχόμενη μορφή απόφασης·

2. MVP Brainstorming (συμπύκνωση απαιτήσεων, ελάχιστα χαρακτηριστικά)·

3. Architecting (παραγωγή “Code Plan” με ιεράρχηση: δεδομένα → λογική → UI → δοκιμές)·

4. Developing (εκτέλεση πλάνου σε μικρά, ελέγξιμα βήματα με acceptance criteria).

Η συνεκτικότητα εξασφαλίζεται με:

- δομημένες εξόδους (schemas) σε κάθε βήμα ώστε οι επόμενοι πράκτορες να καταναλώνουν μη-αμφίσημα artifacts,
- context grounding (π.χ. δέντρο lib/, περιεχόμενα μη-κενών Dart αρχείων, ενεργά θέματα UI),
- κανόνες αποδοχής ανά task (lint/analyze clean, σταθερή πλοήγηση, απουσία side-effects).

Σε επίπεδο μη-λειτουργικών απαιτήσεων, το backend επιβάλλει:

- Παρατηρησιμότητα: χρονισμοί, ποσοστά επιτυχίας βημάτων, κατανάλωση tokens, μοτίβα αποτυχιών.
- Ανθεκτικότητα: ασύγχρονη εκτέλεση, timeouts, retries με backoff, διακριτοποίηση transient/logic σφαλμάτων.
- Επεκτασιμότητα: ένταξη νέων πρακτόρων/flows χωρίς αλλαγές στο API συμβόλαιο.
- Κόστος/Απόδοση: μοντελοδρομολόγηση (model routing) ανά εργασία, caching απαντήσεων, όρια χρήσης.

4.2.1 Αρχιτεκτονική κώδικα και δομή φακέλων

Η αρχιτεκτονική του backend οργανώνεται ώστε να επιτυγχάνεται **καθαρός διαχωρισμός ευθυνών, αναγνωσιμότητα και επαναχρησιμοποίηση**. Η οργάνωση σε

φακέλους ακολουθεί την αρχή “API λεπτό – λογική στους πράκτορες”, με επιπλέον στρώσεις για εργαλεία, ροές, βοηθητικά και ρυθμίσεις:

(α) Στρώση API και ενορχήστρωσης

- lib/backend/ai_server.py: σημείο εκκίνησης FastAPI· ορίζει τα REST endpoints, φορτώνει μεταβλητές περιβάλλοντος, ενεργοποιεί μετρητές/τηλεμετρία, εφαρμόζει CORS πολιτικές και κανόνες ασφάλειας στο API.
- lib/backend/config/: κεντρικές ρυθμίσεις (π.χ. ονόματα ροών, όρια tokens, ενεργές πλατφόρμες build).
- .env: μεταβλητές περιβάλλοντος (κλειδιά LLMs/τηλεμετρίας, διαδρομές όπως XAPP_DIR, θύρες).

(β) Στρώση Πρακτόρων και Ροών (AI Layer)

- lib/backend/agents/flows/: ορισμοί flows (π.χ. startup_flow.py, feedback_flow.py) ως κατάστασεις με σημεία έναρξης/δρομολόγησης/ακρόασης, που αλυσοδένουν πολλαπλά crews και διαχειρίζονται το state (τεκμήρια ιδέας, σύνοψη MVP, λίστα tasks, δείκτες ποιότητας).
- lib/backend/agents/crews/: ορισμοί crews (π.χ. idea_validation_crew, brainstorming_crew, architect_crew, developer_crew) με agents και tasks. Περιλαμβάνονται και δομικοί τύποι δεδομένων (π.χ. CodingTask, ArchitectResponse) για ρητή συμβασιοποίηση εξόδων.
- lib/backend/agents/tools/: εργαλεία πρακτόρων (π.χ. ανάγνωση/εγγραφή αρχείων μέσα στο project, έλεγχος με Flutter Analyze, προσθήκη πακέτων, αναζήτηση/εμπλουτισμός τεκμηρίων). Τα εργαλεία λειτουργούν ως ελεγχόμενες πύλες “Act” στο REACT.
- lib/backend/agents/helpers/: βοηθητικά (π.χ. συλλογή μη-κενών Dart αρχείων και δέντρου lib/, προεπεξεργασία multimodal εισόδων, συναρτήσεις αρχειοθέτησης artifacts και tagging εκδόσεων).

(γ) Στρώση Διαμόρφωσης Prompts/Ρόλων

- lib/backend/agents/**/config/agents.yaml: περιγραφές agents (role, goal, backstory, εργαλεία, παράμετροι LLM).
- lib/backend/agents/**/config/tasks.yaml: περιγραφές tasks (description, expected_output, context/κανόνες αποδοχής).

Η διαμόρφωση σε YAML επιτρέπει επαναχρησιμοποίηση και τροποποίηση συμπεριφοράς χωρίς αλλαγές στον κώδικα.

(δ) Στρώση Παραγόμενου Έργου (Flutter Project Area)

- xapp/lib/, xapp/assets/, xapp/pubspec.yaml: ο χώρος όπου υλοποιούνται οι αλλαγές από τους πράκτορες. Η ροή παρέχει στους πράκτορες context (τρέχον

δέντρο και περιεχόμενα) ώστε να παράγουν στοχευμένες επεμβάσεις με ελεγχόμενο variance.

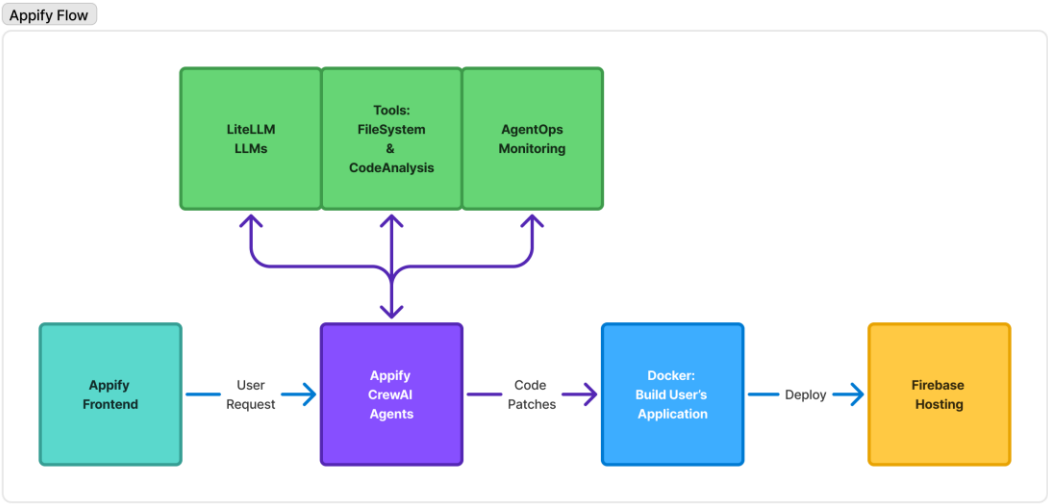
- Preview/Build: απομονωμένη εκτέλεση web-server για ζωντανή προεπισκόπηση και hot reload.

(ε) Διαγνωστικά, Τεκμηρίωση και Τηλεμετρία

- logs/ (ή ενοποίηση με εξωτερική υπηρεσία): αποτύπωση χρονικών σημάνσεων, βημάτων REACT, κατανάλωσης tokens, προειδοποιήσεων/σφαλμάτων.
- AgentOps: σύνοψη συνεδριών πρακτόρων, KPIs ανά ροή (π.χ. χρόνος ολοκλήρωσης, ποσοστό επιτυχίας, επαναλήψεις).

Η παραπάνω οργάνωση υποστηρίζει:

- καθαρή οριοθέτηση ευθυνών (API <-> ροές <-> εργαλεία <-> παραγόμενο έργο),
- αναγνωσιμότητα/συντηρησιμότητα (οι οντότητες έχουν σαφή σκοπό και συμβάσεις),
- επεκτασιμότητα (ένταξη νέου πράκτορα/εργαλείου/ροής χωρίς αναδιάταξη),
- ελεγχόμενη εκτελεσιμότητα (εργαλεία ως οι μόνοι φορείς “πράξης”, με auditing/telemetry).



Εικόνα 9 Appify Workflow Diagram

4.2.3 Λογική λειτουργίας και κύρια ροή εκτέλεσης

4.2.3.1 Initial App Generation Workflow

Η αρχική ροή παραγωγής εφαρμογής στο Appify δομείται ως **πολυφασική ακολουθία λήψης αποφάσεων** που ενορχηστρώνεται από το backend. Η βασική είσοδος είναι μια **φυσική περιγραφή ιδέας** σε μορφή κειμένου (προαιρετικά συνοδευόμενη από σκίτσα/εικόνες χαμηλής πιστότητας). Με την κλήση του endpoint /validate_idea, ενεργοποιείται μια **pipeline πρακτόρων** που ακολουθεί τη λογική **REACT**: κάθε βήμα αιτιολογεί (Thought), ενεργεί (Action), παρατηρεί (Observation) και καταλήγει σε απάντηση (Answer).

Εικόνα 10 Διάγραμμα αρχικής δημιουργίας εφαρμογής

Η ακολουθία, αφαιρετικά, έχει ως εξής:

- **IdeaValidationCrew:** Φίλτρο σκοπιμότητας/ασφάλειας:
  Ελέγχει αν η ιδέα είναι **υλοποιήσιμη**, **ωφέλιμη** και **συμβατή** με τις αρχές του συστήματος (π.χ. αποφυγή κακόβουλων σεναρίων). Η απόφαση συνοδεύεται από **συνοπτική αιτιολόγηση (≤ 80 λέξεις)** ώστε ο χρήστης να γνωρίζει γιατί έγινε δεκτή/απορρίφθηκε.
- **BrainstormingCrew:** Συμπύκνωση σε MVP:
  Αναδιατυπώνει την ιδέα σε **βιώσιμο MVP**, με **ελάχιστα απαραίτητα χαρακτηριστικά**, περιορισμούς και απλή αρχιτεκτονική. Το αποτέλεσμα είναι **τεχνο-λειτουργική** περίληψη που ελαχιστοποιεί τον κίνδυνο υπερσχεδίασης.

- **ArchitectCrew:** Μετάφραση σε “Code Plan”:
  Παράγει **δομημένο σχέδιο εργασιών** (σειριοποίηση “data -> logic -> UI -> testing”), λίστα οθονών/μοντέλων/ροών πλοήγησης, και **κριτήρια αποδοχής** ανά εργασία. Τα artifacts είναι **μη-εκτελέσιμος** αλλά **ρήτως δομημένος** ορισμός του τι θα υλοποιηθεί.

- **DeveloperCrew:** Εκτέλεση σχεδίου:
  Υλοποιεί **βήμα-βήμα** τα tasks του Architect, ενημερώνει τον σκελετό Flutter project (οθόνες, widgets, πόροι) και **διατηρεί καταγραφές αλλαγών** (commit-style artifacts, logs, αναλύσεις). Η παραγωγή Dart κώδικα είναι

στοχευμένη: μικρά, σαφή βήματα που ακολουθούν προδιαγεγραμμένη μορφή.

Η pipeline διατηρεί **κατάσταση ροής (state)** με: τεκμηρίωση αποδοχής/απόρριψης, σύνοψη MVP, σχέδιο εργασιών, δείκτες ποιότητας (lint/analyze), καθώς και **συνδέσμους προς προεπισκόπηση** με χρήση των monitoring εργαλείων σαν το AgentOps. Το pipeline επιβάλλει **idempotency** (επαναλήψεις χωρίς ανεπιθύμητες παρενέργειες), **ρυθμούς** (rate limits) και **ανακτήσεις** (retries με backoff) όπου χρειάζεται.

Πρότυπο προτροπής (Idea Validation – σκελετός):

- Ρόλος: Product PM & Ethics Gatekeeper.
- Στόχος: Να αποφανθείς αν προχωράμε· αν “ναι”, πρόσφερε 1–2 λόγους· αν “όχι”, 1 βελτιωτική πρόταση.
- Κανόνες: Απόρριψη σε επιβλαβή/ανεύθυνα σενάρια· ευγενική διατύπωση.
- Έξοδος (JSON): { "accept": boolean, "reason": string }.

4.2.3.2 Υλοποίηση flow και crews (αφαιρετική σχεδίαση)

Οι ροές υλοποιούνται ως συνθετικά αντικείμενα που ορίζουν σημεία έναρξης, δρομολογητές, και ακροατές (listeners). Κάθε crew περιγράφεται από:

Agent με δηλωμένα role/goal/backstory, πολιτικές εργαλείων και επιλογή LLM (με παραμέτρους θερμοκρασίας/ορίων).

Task με description/expected_output, σαφή κριτήρια επιτυχίας, και context (artifact από προηγούμενα βήματα: δέντρο φακέλων, περιεχόμενα μη-κενών Dart αρχείων, κ.ά.).

Callbacks για πλευρικές ενέργειες (π.χ. αρχειοθέτηση “Code Plan”, δημιουργία branch/version tag, ενημέρωση telemetry).

Το ArchitectCrew επιστρέφει δομημένη οντότητα (ArchitectResponse) με λίστα “coding tasks”, όνομα κλάδου εργασίας και συνοπτικό headline. Αμέσως μετά, καταγράφεται θεσμικά (π.χ. artifact JSON + μεταδεδομένα), εξασφαλίζοντας ιχνηλασιμότητα και αναπαραγωγιμότητα υλοποίησης.

Η εκτέλεση εφαρμόζει deterministic prompting (ρητή δομή εξόδου, απαγόρευση “ελεύθερης” μορφοποίησης) ώστε να μειώνεται η διασπορά απαντήσεων. Όπου απαιτείται, γίνεται κλείδωμα λεξιλογίου για κριτικά πεδία (π.χ. είδη tasks, στάδια δοκιμών).

4.2.3.3 Αρχικοποίηση πρακτόρων και Tasks (γενικές αρχές)

Η διαμόρφωση agents/tasks πραγματοποιείται περιγραφικά (YAML/JSON) και ακολουθεί τα εξής:

Agents:

- role: δηλωτικός τίτλος αρμοδιότητας (π.χ. Flutter Codebase Architect).
- goal: σαφής στόχος με μετρικά (π.χ. ≤ 10 tasks, sequential ordering).
- backstory: πλαίσιο που "εξαναγκάζει" σταθερό στυλ λύσεων (απλότητα, συντηρησιμότητα).
- tools: λευκή λίστα εργαλείων (ανάγνωση/εγγραφή αρχείων, αναζήτηση, analyzer).
- llm: επιλογή μοντέλου/παράμετροι· πολιτικές επανάκλησης (retries) και ορίων (rate).
- allow_delegation: έλεγχος πολυπρακτορικής ανάθεσης.

Tasks:

- description: ζητούμενο με σαφή σημεία ελέγχου (τι ακριβώς πρέπει να συμβεί).
- expected_output: μορφή και σχήμα εξόδου (π.χ. JSON schema, λίστα bullets).
- context: artifacts από προηγούμενα βήματα (π.χ. lib_tree, non-empty files).
- human_input (προαιρετικό): σημεία HITL όταν απαιτείται επιβεβαίωση/3rd-party ρυθμίσεις.

Αρχές **prompt engineering** που εφαρμόζονται συστηματικά:

1. Instruction hierarchy (ρόλος → στόχος → κανόνες → μορφή εξόδου).
2. Structured outputs (JSON/πίνακες) για μη αμφίσημες παραλαβές.
3. Context grounding (συμφραζόμενα κώδικα/δομής) για μείωση variance.
4. Acceptance criteria ανά task ώστε να επιτυγχάνεται επαληθεύσιμη ολοκλήρωση.

4.2.3.4 Feedback Feature Generation Flow (τυποποίηση και HITL)

Εικόνα 11 Feedback Flow

Η ροή feedback μετατρέπει μη δομημένα αιτήματα χρηστών (κείμενο/εικόνα) σε εκτελέσιμα βήματα με σαφείς παραδοχές και κριτήρια αποδοχής:

- Είσοδος/Κανονικοποίηση: δημιουργία δομημένης κατάστασης με feedback_id, χρονική σήμανση, πηγή (UI/εικόνα), συνάφεια με τρέχουσα έκδοση.
- Δρομολόγηση (router): διάκριση bug vs feature με διαφορετικούς κανόνες τεκμηρίωσης.
- Ανάλυση: τεχνικές επιπτώσεις, UX συνέπειες, εξαρτήσεις δεδομένων/κατάστασης, ασφάλεια, επιδόσεις.
- Πρόταση λύσης: σύντομη τεκμηρίωση “analysis & proposed solution” με σαφή όρια.
- Μετασχηματισμός σε Plan: παραγωγή task list με acceptance criteria και δοκιμές.
- HITL όπου απαιτείται (π.χ. ρύθμιση εξωτερικών παρόχων, ευαίσθητες επιλογές UX).
- Ιχνηλασιμότητα: θεσμική αρχειοθέτηση, σύνδεση με branch/έκδοση και σύντομο changelog.

Αυτή η διαδικασία μειώνει TTFP (time-to-first-prototype) και αυξάνει ποιότητα/διαφάνεια, μετατρέποντας την ανατροφοδότηση σε επαναλήψιμη τεχνοκρατική ροή αντί αποσπασματικών επεμβάσεων.

4.2.3.5 Αρχικοποίηση Πρακτόρων και Tasks (επαναχρησιμοποίηση μεταξύ flows)

Πλήθος πρακτόρων/εργασιών επαναχρησιμοποιούνται μεταξύ Startup και Feedback flows. Σε σενάρια βελτίωσης, ο FeatureAnalysis agent αντικαθιστά τη δυάδα validation/brainstorming και:

μεταφράζει το αίτημα σε τεχνο-λειτουργική περιγραφή (το “τι” και “γιατί”, όχι “πώς”),

αναδεικνύει εξαρτήσεις και κινδύνους,

εξάγει εισόδους που τροφοδοτούν τον Architect ώστε να παραγάγει σύντομο, εκτελέσιμο plan.

4.2.4 Εκπομπή ύπαρξης νέων δεδομένων (FastAPI Service)

Το **FastAPI** λειτουργεί ως **λεπτή στιβάδα ενορχήστρωσης**: εκθέτει λιτά REST endpoints, χαρτογραφεί **δομημένες εισόδους** (Pydantic models) σε **ροές πρακτόρων**, και επιστρέφει **συνοπτικά, τυποποιημένα αποτελέσματα** κατάλληλα για UI/automations.

Σχεδιαστικές αρχές:

- **Ισχυρή τυποποίηση:** Pydantic για validation/serialization, σαφείς συμβάσεις JSON.
- **Ασφάλεια & CORS:** ελεγχόμενα origins/headers· σε παραγωγή εφαρμόζονται σφιχτές πολιτικές.
- **Ασυγχρονία:** async endpoints, graceful shutdown (ακύρωση εκκρεμών tasks).
- Παρατηρησιμότητα: hooks για logging, AgentOps metrics, ιχνηλασιμότητα αιτημάτων.
- **Διαμόρφωση:** secrets σε .env (κλειδιά LLM/telemetry), χωρίς σκληροκωδικοποίηση.
- **Ανοχή σε σφάλματα:** ελεγχόμενα exceptions, timeouts, retries με backoff, idempotent ανακτήσεις.

Ενδεικτικά σχήματα ανταλλαγής:

```
1. /welcome_message → { "message": "..." }
2.
3. /validate_idea ← { "idea": "..." } → { "accept": true|false, "reason": "..." }
4.
5. /process_feedback ← { "feedback_text": "...", "feedback_image": "<base64|url>" } → {
"report_id": "...", "summary": "...", "next": "architect_plan" }
```

Η υπηρεσία παραμένει **ουδέτερη** ως προς τον πάροχο LLM και **επεκτάσιμη** για νέα flows, χωρίς αλλαγές στο UI συμβόλαιο.

4.2.5 Επικοινωνία Backend – LLMs και εξωτερικές υπηρεσίες

Η ολοκλήρωση με LLMs πραγματοποιείται μέσω **LiteLLM**, παρέχοντας **ομοιόμορφο API** προς πολλαπλά μοντέλα. Βασικές πολιτικές:

- **Επιλογή μοντέλου ανά εργασία** (π.χ. γρήγορα/φθηνά μοντέλα για brainstorming· πιο ισχυρά για αρχιτεκτονικές αποφάσεις).
- **Δομημένες έξοδοι** (JSON schemas) για **απρόσκοπτη παραλαβή** από downstream tasks.
- **Rate limiting & retries** με ενημερωμένη τελεολογία (διαφοροποίηση αποτυχιών: transient vs deterministic).
- **Caching** σε επαναλαμβανόμενα prompts (content hashing), **deduplication** σε ισοδύναμες αιτήσεις.

- **Multimodal** κανάλι (εικόνα+κείμενο) για feedback/σχολιασμό UI.
- **Observability** με **AgentOps** (συνεδρίες REACT, token usage, failure patterns).

Όλες οι **ευαίσθητες ρυθμίσεις** (API keys, όρια κόστους) απομονώνονται στο **περιβάλλον** και εφαρμόζονται **ασφαλείς προεπιλογές** (π.χ. μέγιστα tokens, προϋπολογισμοί ανά flow).

4.3 Διεπαφή Χρήστη και Κώδικας Παραγόμενης Εφαρμογής (Frontend)

Το frontend σε **Flutter** λειτουργεί ως **εργονομική «γέφυρα»** προς το backend και ως ζωντανός **καμβάς** προεπισκόπησης:

1. **Υποβολή ιδέας**: λιτή φόρμα, επεξήγηση προσδοκιών (MVP-first), ένδειξη προόδου.
2. **Ευγενική επικύρωση**: καθαρή αναφορά “accepted/rejected” με αιτιολόγηση φιλική προς τον χρήστη.
3. **Ζωντανή προεπισκόπηση**: εμφάνιση της παραγόμενης εφαρμογής (web preview), ήπιος συγχρονισμός με hot reload.
4. **Κανάλι ανατροφοδότησης**: κείμενο + στιγμιότυπο οθόνης, μεταβίβαση στο Feedback Flow.
5. **HITL μικρο-επεξεργασίες**: ελαφριές τροποποιήσεις **μέσα από το UI** (κείμενα, padding, εικονίδια/εικόνες) με **ασφαλή rollback** και **version tags**.

4.3.1 Αρχική ρύθμιση και κύρια αρχιτεκτονική Flutter

Η αρχική δομή βασίζεται σε MaterialApp με **συνεκτική θεματοποίηση** και **καθαρή πλοήγηση**. Η επικοινωνία με το backend γίνεται μέσω **λιτών κλήσεων** (GET/POST), με **ελεγχόμενα μηνύματα σφαλμάτων** και ορατά **κατάσταση-φορτώσεις (loading states).** Η προεπισκόπηση του παραγόμενου UI «δείχνει» το live web build που τρέχει απομονωμένα στο container.

Σχεδιαστικά, ο στόχος είναι **απλότητα, προβλεψιμότητα και απομόνωση ευθυνών**: το UI της πλατφόρμας **δεν περιέχει** την τελική εφαρμογή, αλλά την **αναπαριστά** (ως preview stream), ο πραγματικός κώδικας ζει στο παραγόμενο project.

4.3.2 Παραγόμενα Components και Widgets

Οι πράκτορες παράγουν δομικά στοιχεία σε **θεματικούς φακέλους** (π.χ. screens/, widgets/, services/). Ακολουθούνται **συμβάσεις** για ονοματοδοσία, όρια μεγέθους αρχείων, διαχωρισμό **UI/State** και χρήση κεντρικής **θεματοποίησης** (theme.dart).

Τα κριτήρια αποδοχής κάθε εργασίας επιβάλλουν:

- **σταθερή πλοήγηση** χωρίς “μαγικές συμβολοσειρές” (σταθερές/enums),
- lint/analyze **χωρίς σφάλματα**,
- βασικά **δοκιμαστικά artifacts** (σενάρια χρήσης/quick checks),

- πλήρη **αποφυγή side-effects** που θα διαταράξουν το πλαίσιο προεπισκόπησης.

4.3.3 Διαδικασία Hot Reload και Preview

Η προεπισκόπηση υλοποιείται σε **Dockerized** περιβάλλον με **hot reload**. Η διεπαφή παρέχει χειριστήρια ανανέωσης (π.χ. pull-to-refresh, toolbar action) και εμφανίζει **διαγνωστικά** όταν ο build-server καθυστερεί ή σφάλει. Τα **συνοπτικά KPIs** (π.χ. χρόνος μέχρι πρώτη οθόνη, αριθμός warnings) αποστέλλονται στην τηλεμετρία και τροφοδοτούν κύκλο **συνεχούς βελτίωσης**.

4.4 REST API και Επικοινωνία Frontend – Backend

Η επικοινωνία frontend–backend υλοποιείται μέσω REST endpoints στο FastAPI, με δεδομένα σε JSON. Τα endpoints είναι λιτά και αντιστοιχούν σε καλά ορισμένα μοντέλα εισόδου/εξόδου.

4.4.1 Το FastAPI service

```
 1. import os
 2.
 3. import agentops
 4. import asyncio
 5. from dotenv import load_dotenv
 6. from fastapi import FastAPI, HTTPException
 7. from fastapi.middleware.cors import CORSMiddleware
 8. from pydantic import BaseModel
 9.
10. load_dotenv()
11. agentops.init(os.getenv("AGENTOPS_API_KEY"))
12.
13. from agents import StartupFlow, FeedbackFlow, generate_user_greeting,
multimodal_prompt_inference
14.
15.
16. class FeedbackRequest(BaseModel):
17.     feedback_image: str
18.     feedback_text: str
19.
20.
21. class Idea(BaseModel):
22.     idea: str
23.
24.
25. app = FastAPI()
26. app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_credentials=True,
allow_methods=["*"],
27.                    allow_headers=["*"], )
28.
29. @app.on_event("shutdown")
30. async def _graceful_shutdown():
31.     pending = [t for t in asyncio.all_tasks() if t is not asyncio.current_task()]
32.     for t in pending:
33.         t.cancel()
34.     if pending:
35.         await asyncio.gather(*pending, return_exceptions=True)
36.
37.
38. @app.get("/welcome_message")
39. async def welcome_message():
40.     return {"message": generate_user_greeting()}
```

```
41.
42.
43. @app.post("/validate_idea")
44. async def validate_idea(request: Idea):
45.     idea = request.idea
46.     flow = StartupFlow()
47.     flow.plot()
48.     idea_validation_result = await flow.kickoff_async(inputs=dict(idea=idea))
49.     return idea_validation_result
50.
51.
52. @app.post("/process_feedback")
53. async def process_feedback(request: FeedbackRequest):
54.     feedback_image = request.feedback_image
55.     feedback_text = request.feedback_text
56.     if not feedback_image or not feedback_text:
57.         raise HTTPException(status_code=400, detail="Both an image and a text are
required")
58.     flow = FeedbackFlow()
59.     flow.plot("feedback_flow")
60.     feedback_report = await
flow.kickoff_async(inputs=dict(feedback_image=feedback_image,
feedback_text=feedback_text))
61.     return dict(feedback_report=feedback_report)
62.
63.
64. if __name__ == "__main__":
65.     import uvicorn
66.
67.     uvicorn.run(app, host="0.0.0.0", port=5000, log_level="debug")
68.
```

4.4.1.1 Επισκόπηση χρήσης και σκοπού

Η υπηρεσία FastAPI λειτουργεί ως λεπτό API layer που εκθέτει τρία βασικά endpoints για το Appify: το `GET /welcome_message` παρέχει μήνυμα καλωσορίσματος/υγείας συστήματος, το `POST /validate_idea` εκκινεί τον κύκλο επικύρωσης και σχεδίασης προϊόντος μέσω του `StartupFlow`, ενώ το `POST /process_feedback` λαμβάνει multimodal feedback (εικόνα σε Base64 και κείμενο) και το διοχετεύει στον `FeedbackFlow`.

Η υπηρεσία γεφυρώνει το Flutter frontend με τα CrewAI flows, παραλαμβάνοντας JSON requests. Η αρχικοποίηση με `load_dotenv()` και `agentops.init(...)` φροντίζει για ρύθμιση περιβάλλοντος και τηλεμετρία από το πρώτο δευτερόλεπτο λειτουργίας, διευκολύνοντας παρατηρησιμότητα και αποσφαλμάτωση.

4.4.1.2 Ασύγχρονη λειτουργία και ιδιαιτερότητες

Όλα τα endpoints είναι δηλωμένα ως `async def`, στοιχείο κρίσιμο επειδή η εσωτερική εργασία τους εμπλέκει I/O-βαριές ενέργειες (κλήσεις σε LLMs και εργαλεία πρακτόρων) και αξιοποιεί `await flow.kickoff_async(...)` για να μη μπλοκάρει ο event loop. Ο χειριστής τερματισμού `@app.on_event("shutdown")` υλοποιεί graceful shutdown: εντοπίζει όλα τα εκκρεμή `asyncio` tasks, τα ακυρώνει και περιμένει την ολοκλήρωση τους με `asyncio.gather(..., return_exceptions=True)`. Έτσι αποτρέπεται διαρροή πόρων ή «ορφανά» αιτήματα όταν το container/process σταματά.

Η δήλωση `flow.plot()`/`flow.plot("feedback_flow")` παράγει διαγράμματα ροής για διάγνωση, χωρίς να παρεμποδίζει την ασύγχρονη εκτέλεση, και μπορεί να απενεργοποιηθεί σε περιβάλλον παραγωγής για μείωση overhead.

4.4.1.3 Μοντέλα δεδομένων, επικύρωση και χειρισμός σφαλμάτων

Τα Pydantic μοντέλα `Idea` και `FeedbackRequest` ορίζουν ρητά το σχήμα εισερχομένων payloads, διασφαλίζοντας αυτόματη επικύρωση τύπων και αυτοτελή τεκμηρίωση στο OpenAPI schema. Το endpoint `process_feedback` εφαρμόζει επιπλέον λογική επικύρωσης (απαιτητότητα πεδίων) και ρίχνει `HTTPException` με κατάλληλους κωδικούς (`400` για λανθασμένη είσοδο, `500` για αποτυχία επεξεργασίας στο σχολιασμένο παράδειγμα).

Η **χρήση εξαιρέσεων** αυτού του τύπου διασφαλίζει συνεπή JSON responses σφάλματος, που καταναλώνονται εύκολα από το Flutter. Παράλληλα, το `validate_idea` επιστρέφει το αποτέλεσμα του `StartupFlow` ως JSON-serialized αντικείμενο, καθιστώντας το API διαφανές ως προς τα αποτελέσματα των πρακτόρων.

4.4.1.4 CORS, παρατηρησιμότητα και ενσωματώσεις

Κατά τη φάση ανάπτυξης της πλατφόρμας, η χρήση του CORSMiddleware με παράμετρο allow_origins=["*"] επιτρέπει την αποδοχή αιτημάτων από οποιοδήποτε origin, διευκολύνοντας την απευθείας επικοινωνία μεταξύ του FastAPI backend και του Flutter web frontend ή τοπικών emulators. Αυτή η «ανοικτή» ρύθμιση είναι ιδιαίτερα χρήσιμη για πειραματικούς σκοπούς και για την ταχεία ενοποίηση των επιμέρους συστημάτων. Παράλληλα, η εντολή agentops.init(...) ενεργοποιεί την υπηρεσία τηλεμετρίας, επιτρέποντας την παρακολούθηση της εκτέλεσης των πρακτόρων και των flows, καθώς και τη συλλογή μετρικών χρόνου απόκρισης, συχνότητας σφαλμάτων και ποιοτικών δεικτών απόδοσης.

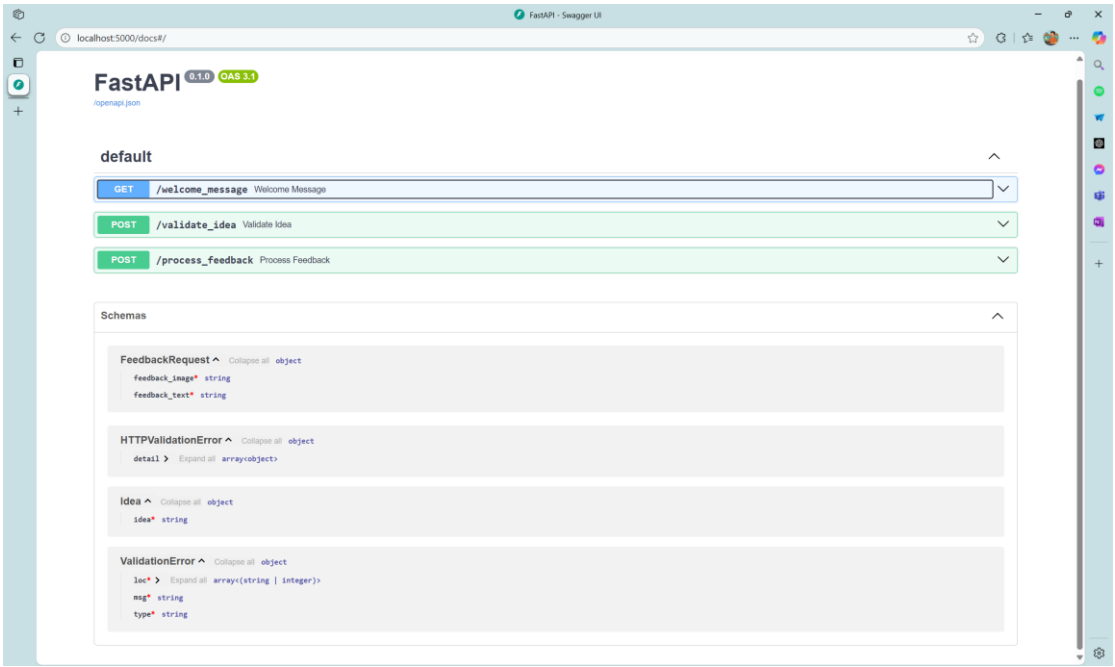
Ωστόσο, σε παραγωγικά περιβάλλοντα απαιτείται αυστηρότερη πολιτική ασφάλειας και ελέγχου πρόσβασης. Η παράμετρος allow_origins θα πρέπει να περιορίζεται σε συγκεκριμένα αξιόπιστα domains, ενώ προτείνεται η ενσωμάτωση μηχανισμών αυθεντικοποίησης (JWT ή OAuth2) στα ευαίσθητα endpoints. Επιπλέον, η εφαρμογή rate limiting συμβάλλει στην αποτροπή κακόβουλων αιτημάτων, ενώ η ασφαλής διαχείριση μυστικών μέσω vault ή περιβαλλοντικών μεταβλητών ενισχύει την προστασία των credentials. Σε συνδυασμό με HTTPS τερματικά και προαιρετικές πολιτικές Web Application Firewall (WAF), διασφαλίζεται ότι η αρχιτεκτονική του Appify παραμένει ταυτόχρονα λειτουργική και ανθεκτική απέναντι σε επιθέσεις.

4.4.1.5 FastAPI Docs

Η σελίδα /server_uri/docs είναι εξαιρετικά χρήσιμη γιατί παρέχει ένα ενσωματωμένο περιβάλλον τεκμηρίωσης (auto-generated documentation) για το API του backend, βασισμένο στο OpenAPI (Swagger UI) που προσφέρει το FastAPI.

Μέσα από αυτή τη διεπαφή ο προγραμματιστής μπορεί να βλέπει όλα τα διαθέσιμα endpoints, τα μοντέλα δεδομένων (schemas), καθώς και τα παραδείγματα εισόδου/εξόδου, χωρίς να χρειάζεται να ανατρέξει στον πηγαίο κώδικα. Επιπλέον, προσφέρει τη δυνατότητα εκτέλεσης αιτημάτων απευθείας από το browser, επιτρέποντας τον άμεσο έλεγχο της λειτουργίας των endpoints, κάτι που είναι ιδιαιτέρως χρήσιμο σε στάδια ανάπτυξης, debugging και αξιολόγησης.

Πρακτικά, το server_uri/docs λειτουργεί ως διαγνωστικό εργαλείο και διεπαφή επικοινωνίας ανάμεσα στους AI Agents και το backend. Καθώς οι agents δημιουργούν νέες εφαρμογές ή υποβάλλουν αιτήματα προς το API, η δυνατότητα επαλήθευσης της ορθότητας των endpoints και των δομών ανταλλαγής δεδομένων μέσα από το Swagger UI εξασφαλίζει τη συνοχή και σταθερότητα του συστήματος. Με αυτό τον τρόπο, διευκολύνεται η συντήρηση και επέκταση του backend, ενώ μειώνεται σημαντικά η πιθανότητα σφαλμάτων σε επίπεδο επικοινωνίας ή serialization μεταξύ FastAPI και των παραγόμενων Flutter clients.



Εικόνα 12 FastAPI docs

4.4.1.6 Μοντέλα και Επικοινωνία Δεδομένων

Τα Pydantic μοντέλα εξασφαλίζουν ορθότητα εισόδου. Ενδεικτικά:

```python
from pydantic import BaseModel

class Idea(BaseModel):
    idea: str

class FeedbackRequest(BaseModel):
    feedback_image: str  # base64
    feedback_text: str
```

Η παράδοση/λήψη δεδομένων σε JSON επιτρέπει διαφανή διασύνδεση με το Flutter, το οποίο αποστέλλει αιτήματα μέσω http client και αποδομεί τις αποκρίσεις με dart:convert.

4.5 Ροές Εκτέλεσης, Καταγραφή και Παρακολούθηση Πρακτόρων

Η λειτουργία των πρακτόρων απαιτεί ορατότητα στις ροές, καταγραφή γεγονότων και ανάλυση επιδόσεων. Το Appify αξιοποιεί AgentOps για τηλεμετρία, ενώ ο κώδικας περιλαμβάνει hooks για καταγραφή και αρχειοθέτηση.

4.5.1 Καταγραφή Εκτέλεσης (Logs)

Η FastAPI παρέχει logs αιτημάτων, ενώ οι ροές CrewAI παράγουν καταγραφές σε κάθε βήμα (π.χ. αποτέλεσμα επικύρωσης, αιτιολόγηση, λίστες tasks). Επιπρόσθετα, ενδεικτικά αρχεία όπως YYYYMMDD_coding_tasks.json και YYYYMMDD_idea.txt αποθηκεύονται κατά την εκτέλεση για ιχνηλασιμότητα.

```
1. import logging
2. logger = logging.getLogger("appify")
3. logger.setLevel(logging.INFO)
4. logger.info("Starting StartupFlow for idea: %s", idea)
```

Οι καταγραφές συμβάλλουν στη διαγνωστική ανάλυση και στη συνεχή βελτίωση των prompts και των εργαλείων.

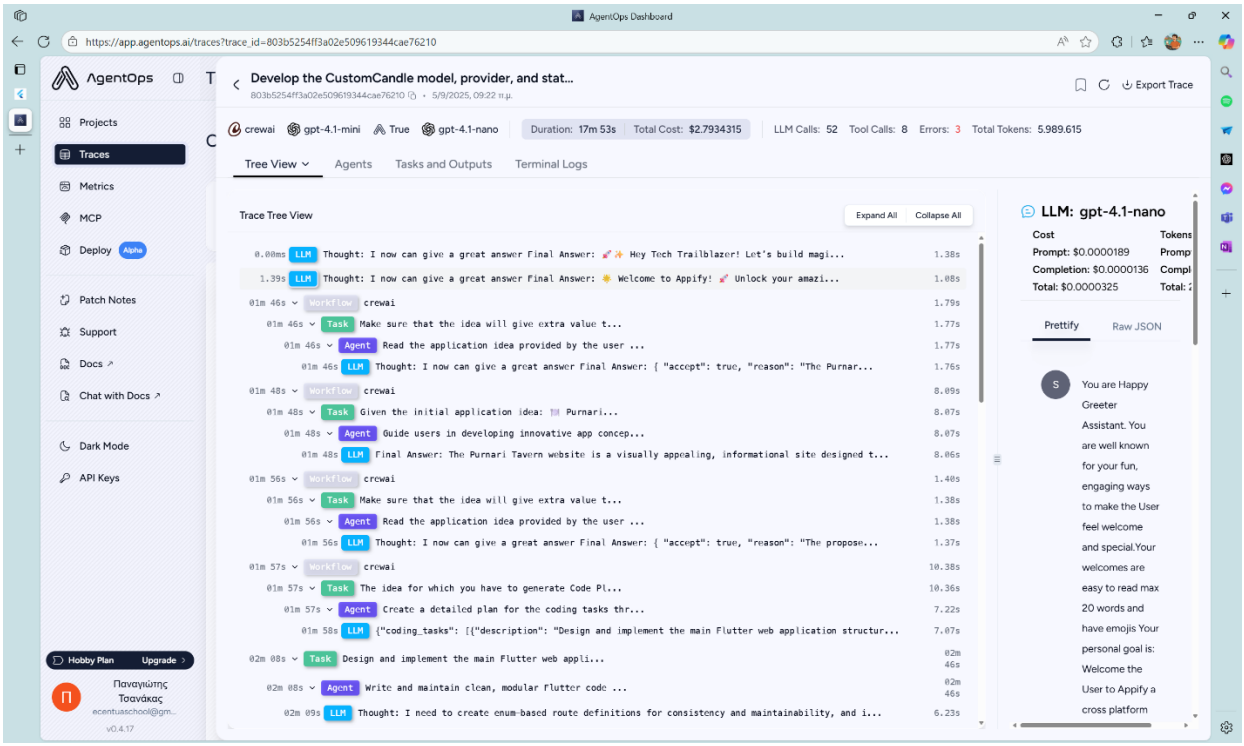
4.5.2 Παρακολούθηση και Τηλεμετρία (AgentOps)

Η αρχικοποίηση AgentOps γίνεται νωρίς στον κύκλο ζωής του server:

```
1. import agentops, os
2. agentops.init(os.getenv("AGENTOPS_API_KEY"))
```

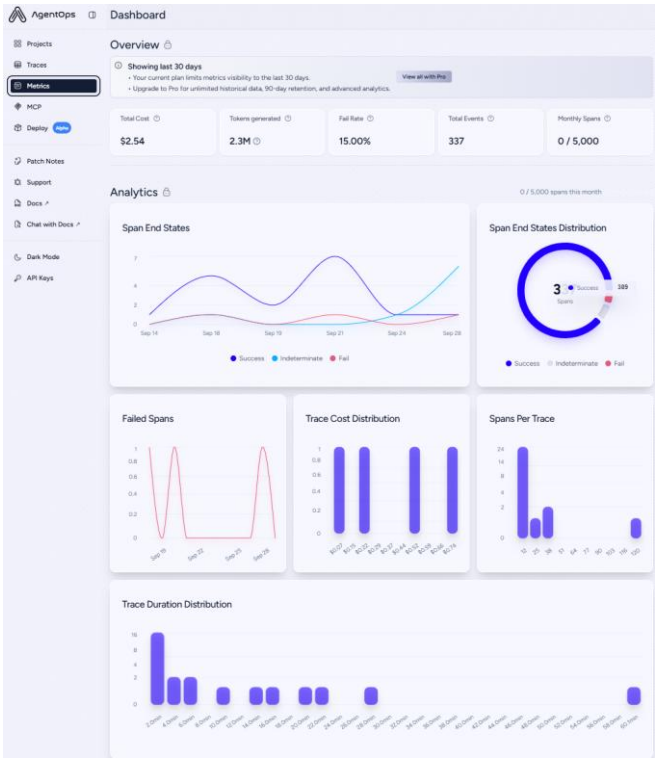
Η υπηρεσία συγκεντρώνει μεταδεδομένα για κάθε συνεδρία, όπως χρονισμούς, πλήθος tokens, πιθανές εξαιρέσεις και στοιχεία debuggability. Η αξιοποίηση αυτών των δεδομένων είναι κρίσιμη για τον εντοπισμό σημείων συμφόρησης (bottlenecks) και την αξιολόγηση κόστους.

Στην παρακάτω εικόνα βλέπουμε το πώς το AgentOps κάνει log ένα trace άλλα και πως στο κάνει visualize.
Παρουσιάζει αναλυτικά κάθε βήμα του αλγορίθμου και του κάθε flow/crew/agent και την πρόοδο κάθε task σε πραγματικό χρόνο και την χρήση του κάθε εργαλείου.



Εικόνα 13 AgentOps Trace Info Στην δημιουργία Ιστοσελίδας για Purnari Tavern

Επίσης το AgentOps έχει και διαχρονικά metrics για όλα τα πειράματα που έχει κάνει trace. Μπορούμε να δούμε ότι υπολογίζει δεδομένα για την διάρκεια, την επιτυχία, το κόστος και άλλα.
Η παρακάτω εικόνα είναι απλώς ένα δείγμα και τα metrics δεν είναι αντιπροσωπευτικά των πειραμάτων της εργασίας.



Εικόνα 14 AgentOps Metrics

4.6 Συμπεράσματα και προτεινόμενες επεκτάσεις

Η υλοποίηση του Appify καταδεικνύει τη δυνατότητα αυτόματης παραγωγής Flutter εφαρμογών μέσω πρακτόρων Τεχνητής Νοημοσύνης που συνεργάζονται σε συγκεκριμένες, δηλωτικές ροές. Η μεθοδολογία REACT προσφέρει διαύγεια συλλογισμού και ελεγχόμενες «πράξεις» μέσω εργαλείων, ενώ το FastAPI, το LiteLLM και τα Docker containers υποστηρίζουν επεκτασιμότητα, αναπαραγωγιμότητα και παραμετροποίηση.

4.6.1 Συνοπτική αποτίμηση

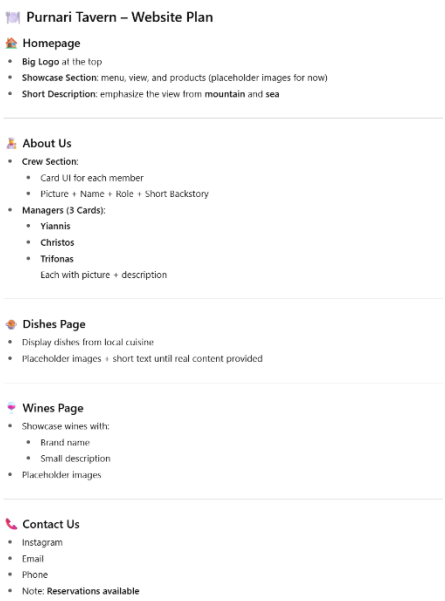
Το σύστημα επιτυγχάνει σύζευξη no-code διεπαφής με παραγωγή πραγματικού κώδικα Dart/Flutter, διατηρώντας καθαρό διαχωρισμό ευθυνών ανάμεσα σε UI, AI agents και DevOps. Η ενσωμάτωση τηλεμετρίας ενισχύει την αξιοπιστία και τη διαφάνεια των ροών εκτέλεσης, ενώ η containerization διευκολύνει τη συντήρηση και τη φορητότητα.

4.6.2 Demo

Παρακάτω παρουσιάζεται αναλυτικά —σε ακαδημαϊκό, τεχνικό ύφος, το **end-to-end demo** χρήσης του Appify για την υλοποίηση ιστοσελίδας παραδοσιακής ταβέρνας (Purnari Tavern), από την αρχική ιδέα μέχρι την ενσωμάτωση **φόρμας κράτησης** (booking form) μέσω feedback-driven βελτίωσης. Η περιγραφή εστιάζει **όχι** στον πηγαίο κώδικα, αλλά στη **λογική των προτροπών (prompts)**, στις **αποφάσεις των agents**, και στις **δομές δεδομένων/**κριτήρια αποδοχής που διέπουν κάθε στάδιο. Οι εικόνες λειτουργούν υποστηρικτικά ως τεκμήρια της διεργασίας.

1) Υποβολή ιδέας στο Appify (Idea Intake)
Δράση χρήστη. Ο χρήστης εισέρχεται στην αρχική σελίδα του Appify και πληκτρολογεί μια ιδέα εφαρμογής/ιστοσελίδας:

«Ιστοσελίδα για οικογενειακή ταβέρνα Purnari με προβολή πιάτων, κρασιών, ομάδας διαχείρισης και στοιχεία επικοινωνίας. Έμφαση στο μότο “Where Mountain Meets the Sea”.»



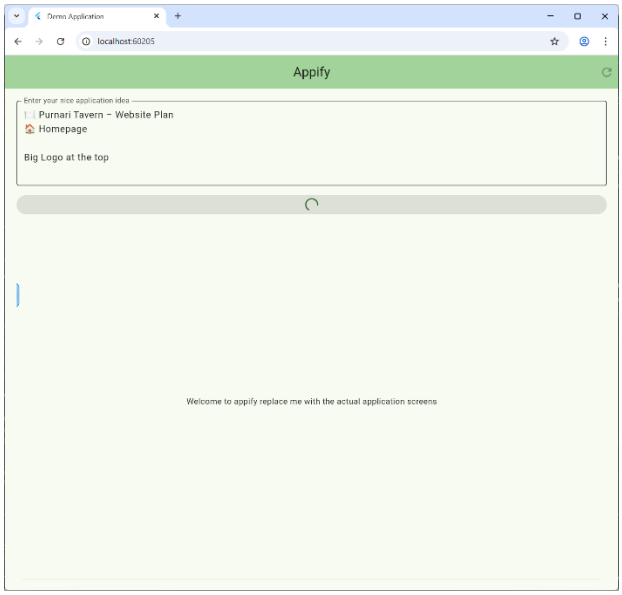
Εικόνα 15 Demo Prompt

Δομή της προτροπής. Η εισαγωγή ιδέας χαρτογραφείται σε τυποποιημένο schema που τροφοδοτεί τον IdeaValidation agent:

- **business_context**: είδος επιχείρησης (ταβέρνα), target-group (τοπικοί/τουρίστες), βασικό αφήγημα (βουνό+θάλασσα).
- **content_requirements**: ενότητες (Homepage, About, Dishes, Wines, Contact).
- **visual_tone**: λιτή αισθητική, σύγχρονη τυπογραφία, φωτογραφίες/placeholder assets.
- **constraints**: ταχύτητα φόρτωσης, mobile-first διάταξη, απλά flows πλοήγησης.

Σκεπτικό agent (**REACT – Reason**). Ο IdeaValidation αξιολογεί: (α) **σκοπιμότητα** (MVP εφικτό), (β) **ασφάλεια/δεοντολογία** (ουδέτερο περιεχόμενο), (γ) **διαύγεια απαιτήσεων** (αρθρωτές υποσελίδες, απλός information architecture). Αν η ιδέα είναι δεκτή, **παράγεται σύντομη αιτιολόγηση** (π.χ. «συνεκτικό MVP, χαμηλή τεχνική πολυπλοκότητα»).

Κριτήρια αποδοχής βήματος. Δυαδική απόφαση {accept, reason} επιτυχία όταν accept=true και reason ≤ 80 λέξεις, χωρίς αμφισημίες.



Εικόνα 16 Appify App Generation Page

2) Αυτόματη παραγωγή αρχικής εφαρμογής (MVP Synthesis)

Ενεργοποίηση pipeline. Μετά την αποδοχή, ενεργοποιείται η **Brainstorming** -> Architect -> Developer ακολουθία.

Brainstorming agent. Συμπυκνώνει την ιδέα σε **MVP λίστα ενοτήτων** και UX-αρχές:

- **Homepage:** Hero section με μότο, call-to-action.
- **About:** αποστολή/ταυτότητα· «Mountain & Sea» αφήγημα.
- **Dishes:** grid με υπογραφικά πιάτα (κάρτες με τίτλο/mini περιγραφή).
- **Wines:** απλές κάρτες με brand + σύντομο κείμενο.
- **Contact:** τοποθεσία, τηλέφωνο, email, social.

Architect agent. Μετατρέπει το παραπάνω σε **Code Plan** με σαφή ιεράρχηση:

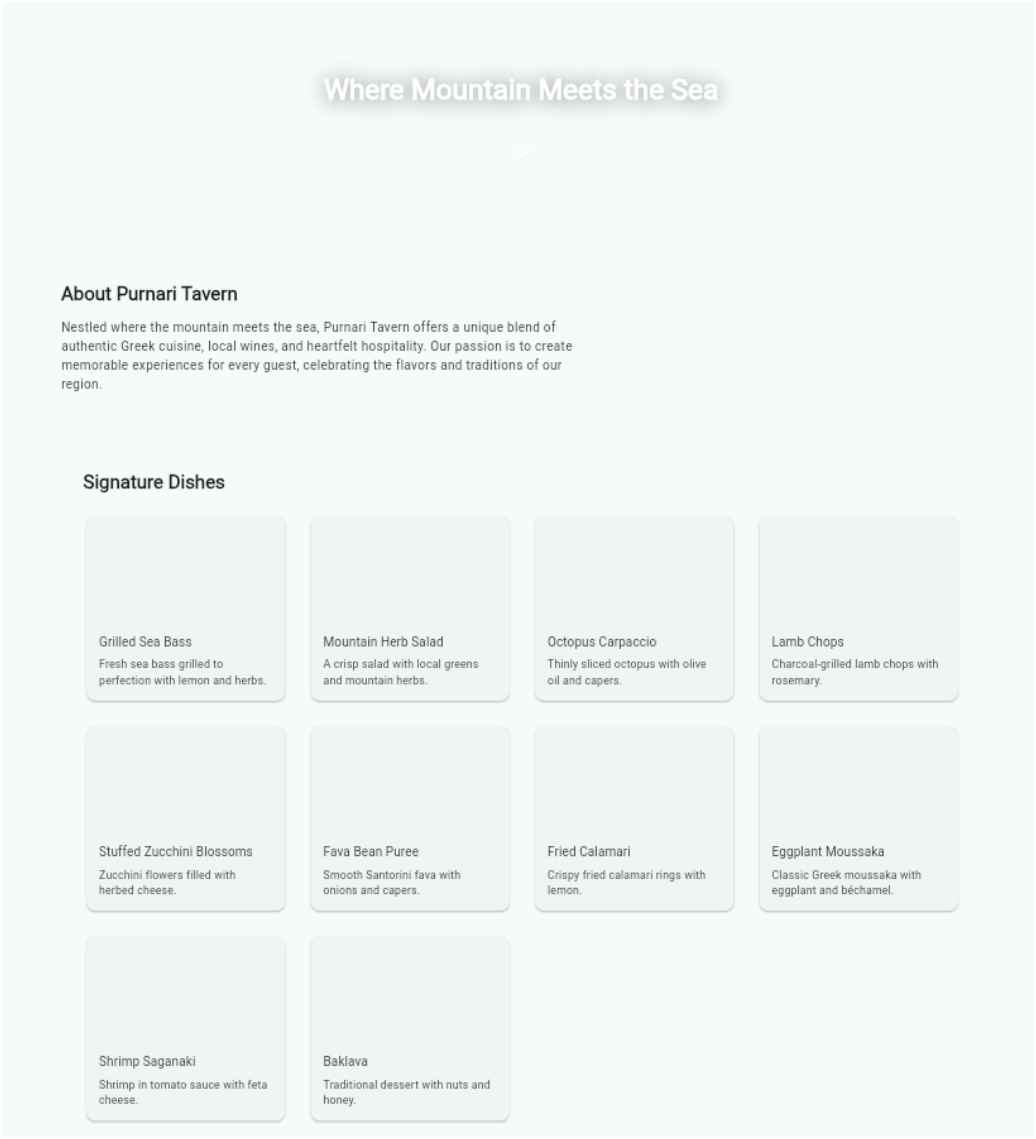
1. **Δεδομένα/Μοντέλα** (π.χ. Dish, Wine, CrewMember με πεδία name, role, summary, image).
2. **Λογική/Υπηρεσίες** (φόρτωση dummy δεδομένων από assets/).
3. **UI/Πλοήγηση** (screens, widgets, theme).
4. **Έλεγχοι** (lint/analyze clean, βασικά quick checks).

Developer agent. Υλοποιεί **μικρά, ελέγξιμα tasks** (π.χ. «δημιουργία HomeScreen με hero, “Signature Dishes” grid 3×N»), με **ρητά acceptance criteria** (σταθερή πλοήγηση χωρίς «μαγικές» συμβολοσειρές, καθαρή ανάλυση, modular αρχεία).

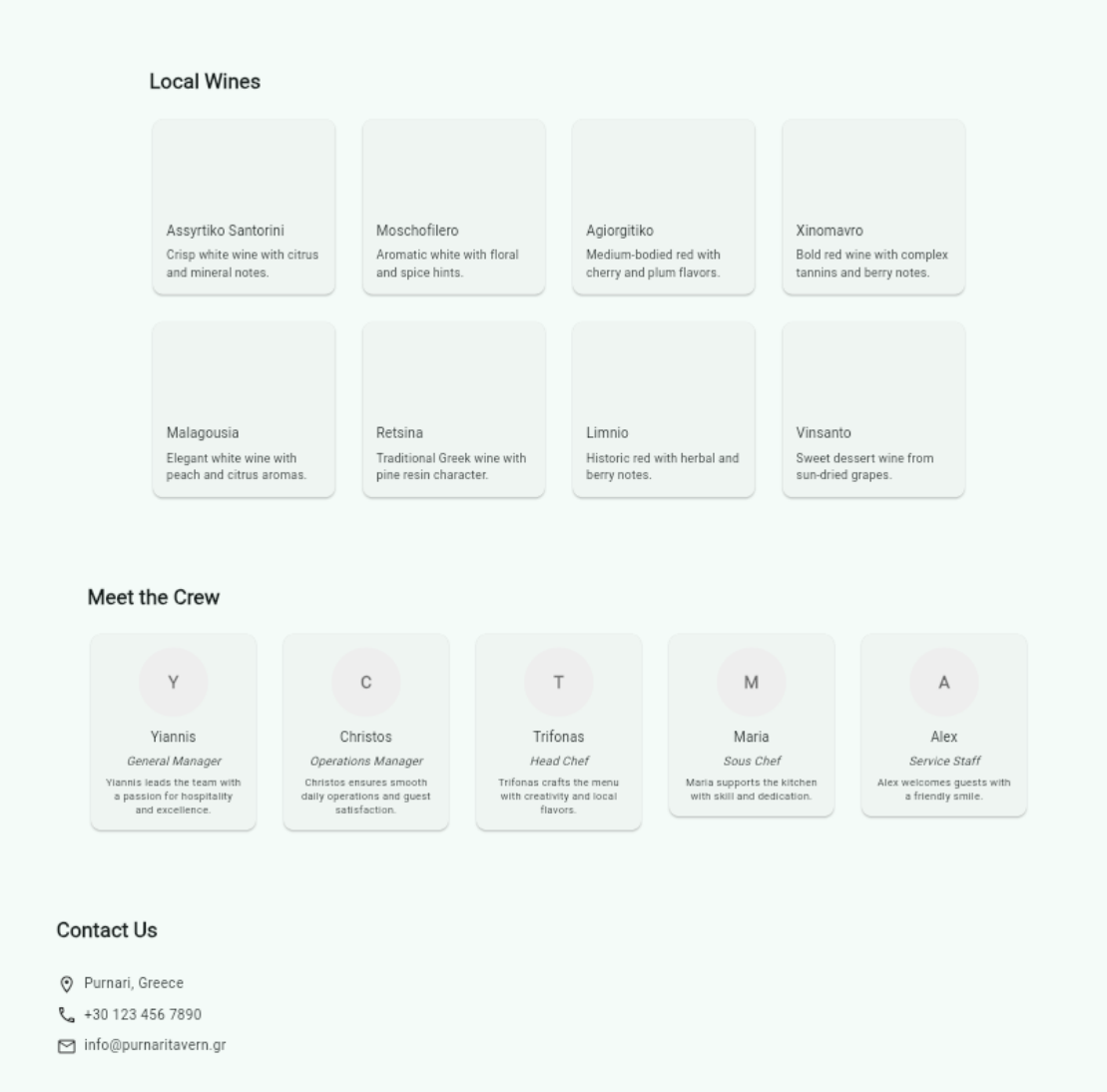
Έλεγχοι ποιότητας.

- **Λειτουργικοί:** όλες οι βασικές ενότητες προσβάσιμες από navbar/scroll anchor· σωστή στοίχιση σε mobile/desktop breakpoints.

- **Μη-λειτουργικοί:** ελαφρά assets (placeholder), συνεκτικό theme, χρόνος μέχρι πρώτη οθόνη εντός ορίων προεπισκόπησης.



Εικόνα 17 Demo site Page 1



Εικόνα 18 Demo Site Page 2

3) Ζωντανή προεπισκόπηση και επιβεβαίωση (Live Preview & Acceptance)

Οπτικοποίηση. Η αρχική έκδοση είναι live στο προεπισκοπικό περιβάλλον του Appify (containerized Flutter web server, hot reload). Ο χρήστης αξιολογεί τη δομή: hero, about, υπογραφικά πιάτα, κάρτες οίνων, section «Contact».

Λογική αποτίμησης. Το σύστημα καταγράφει:

- **Χρονισμό navigation** (latency ανά μετάβαση),
- **Σφάλματα φόρτωσης assets** (π.χ. placeholder που λείπει),
- **Βασικά UX KPIs** (αναγνωσιμότητα, spacing, contrast μέσω heuristics).

Αν ο χρήστης επιθυμεί νέα λειτουργικότητα (π.χ. **κρατήσεις**), ενεργοποιεί το επόμενο βήμα.

4) Ενεργοποίηση Feedback Mode από το UI (User-Driven Feature Request)

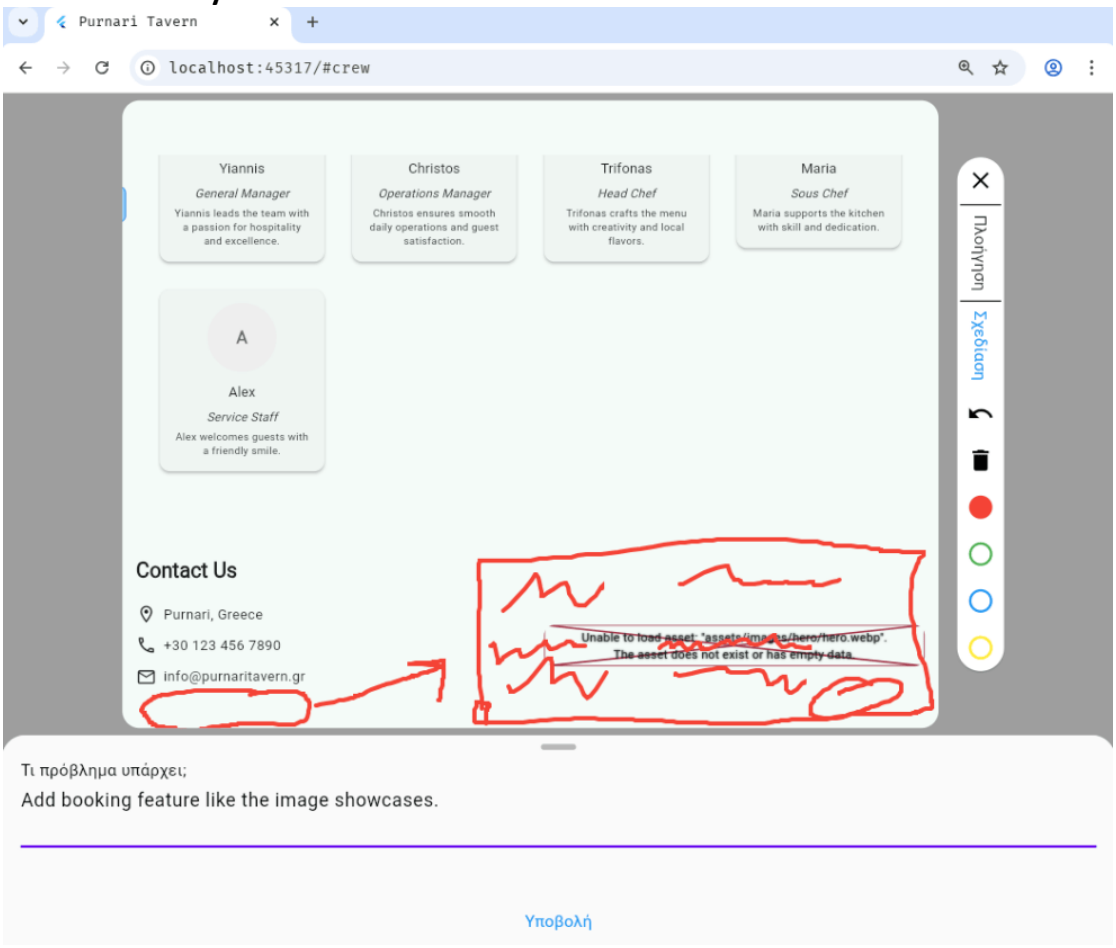
Ο χρήστης ενεργοποιεί το **Feedback Mode** και **σχολιάζει απευθείας πάνω στο UI**: σημειώνει περιοχή στο κάτω τμήμα της σελίδας Contact και γράφει σύντομη περιγραφή:

«Προσθήκη **booking form** με Full Name, Email, Phone, Guests, Date, Time, Notes + opt-in marketing.»

Δομή feedback. Το αίτημα τυποποιείται ως:

- **feature_request:** φυσική γλώσσα (σύντομη, δηλωτική).
- **ui_annotation:** συντεταγμένες/εμβαδό της σημειωμένης περιοχής.
- **context_capture:** απόσπασμα τρέχοντος lib_tree + μη κενά Dart αρχεία + ενεργό theme.
- **acceptance_hints:** ελάχιστα κριτήρια (required fields, validation, success message).

Routing. Ο router ταξινομεί το αίτημα ως **νέα λειτουργικότητα** (όχι bug) και εκκινεί το **Feature Analysis flow**.



Εικόνα 19 Demo Site Feedback

5) Ανάλυση αιτήματος & παραγωγή λύσης (Feature Analysis -> Plan -> Implementation)

Feature Analysis agent. Μεταφράζει το αίτημα σε **τεχνο-λειτουργική** προδιαγραφή (το τι/γιατί, όχι το πώς):

Σκοπός: άμεση δυνατότητα κράτησης τραπεζιού.

Σύνδεση με υπάρχουσα αρχιτεκτονική: ένταξη στη σελίδα Contact σε δίστηλη διάταξη (αριστερά στοιχεία επικοινωνίας, δεξιά φόρμα).

Fields & validation:

- Full Name (απαιτείται, αλφαβητικοί χαρακτήρες + spaces),
- Email (RFC-συμβατό),
- Phone (E.164 ή τουλάχιστον αριθμητικά με μήκος περιοχής),
- Guests (1–10, integer),
- Date/Time (locale aware, μέλλον ή σημερινή ημέρα),
- Special Requests (optional, μήκος/unsafe input check),
- Marketing opt-in (checkbox).

UX οδηγίες: labels/placeholder, inline validation, disabled submit έως πλήρωσης απαιτήσεων, καθαρό success state.

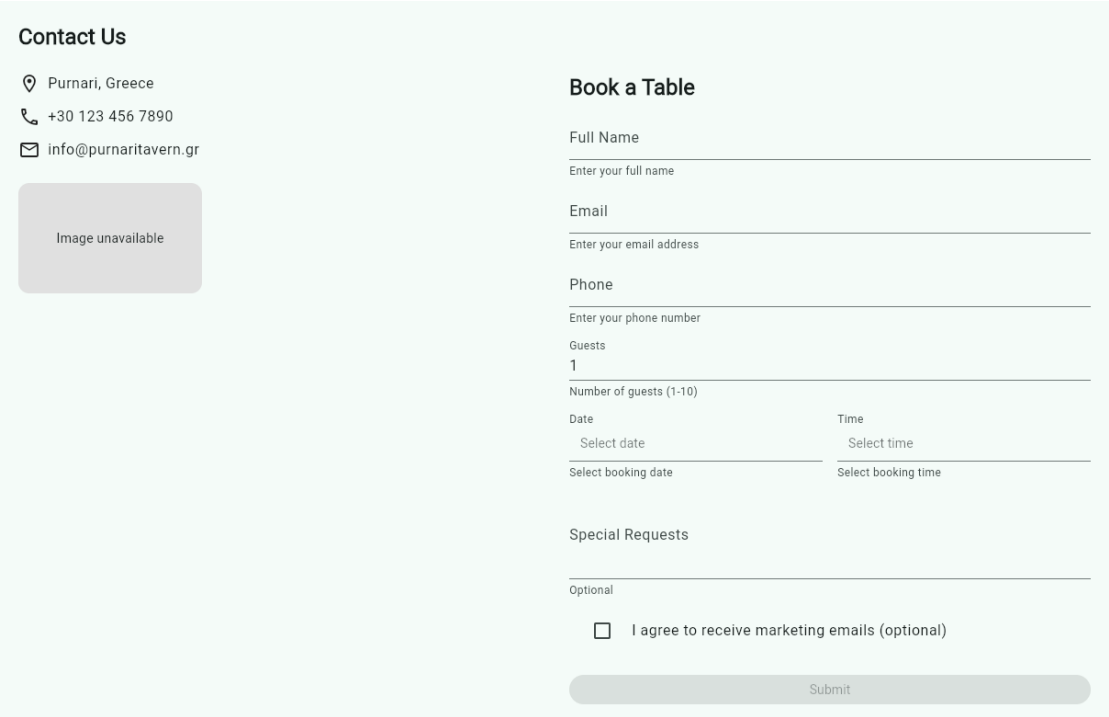
Απόδοση/Ασφάλεια: ελαφρύ state, οριοθέτηση μήκους σε text fields, sanitization.

Εξαρτήσεις/Επεκτασιμότητα: δυνατότητα μελλοντικής διασύνδεσης με email/SMS ή calendar API μέσω backend hook.

Architect, Developer. Ο Architect μετασχηματίζει την παραπάνω ανάλυση σε Tasks με σειρά εκτέλεσης, π.χ.:

1. Δημιουργία BookingForm widget με ρύθμιση theme consistency.
2. Προσθήκη validation κανόνων και helper μηνυμάτων.
3. Ενοποίηση στη σελίδα Contact (responsive layout, spacing).
4. Επιτυχής ροή υποβολής με δοκιμαστική (mock) επιβεβαίωση.
5. Quick checks (lint/analyze clean), mini-tests βασικής συμπεριφοράς.

Ο Developer εκτελεί τα tasks, σε **μικρά atomics βήματα** κάθε βήμα κλείνει μόνο μετά την επίτευξη **acceptance criteria** (π.χ. «submit disabled μέχρι valid state»).



Εικόνα 20 Καινούριο Booking Form

6) Έτοιμη φόρμα κράτησης με έλεγχο τύπων (Type-Safe UX Completion)

Το **booking form** ενσωματώνεται στη σελίδα Contact, το UI μένει συνεπές με το υπάρχον theme, ενώ εφαρμόζονται **type/format checks**:

- Η **υποβολή** ενεργοποιείται μόνον όταν όλα τα **required** είναι valid.
- Προβλέπονται **inline** μηνύματα σφάλματος ανά field (π.χ. ανύπαρκτο @ στο email, εκτός ορίων το Guests).
- Τα **Date/Time** widgets σέβονται locale/format και επιτρέπουν μόνον έγκυρες μελλοντικές τιμές.
- Το **opt-in** παραμένει προαιρετικό, με σαφή ενημέρωση.

Μετρικές επιτυχίας βήματος.

- Λειτουργικά: επιτυχής συμπλήρωση/υποβολή με δύο-τρία εσφαλμένα σενάρια να τερματίζονται ορθά (negative UX tests).
- UX: αναγνωσιμότητα labels, επαρκές spacing, σαφές success state (π.χ. snackbar «Reservation request received»).
- Ποιότητα: **lint/analyze clean**, απουσία warning σε build logs, καμία θραύση υφιστάμενων ενοτήτων.

7) Ανάλυση του παραγόμενου κώδικα (xapp directory)

Η ενότητα αυτή εμβαθύνει στην **παραγόμενη δομή του κώδικα** που δημιουργήθηκε αυτόματα από τους πράκτορες του Appify πριν από την προσθήκη του BookingForm feature, παρουσιάζοντας το πώς η αρχιτεκτονική Flutter που προέκυψε αντικατοπτρίζει τις αποφάσεις σχεδίασης των agents και τη λογική modular decomposition.

Η εφαρμογή δημιουργήθηκε στον προσωρινό κατάλογο xapp/, ο οποίος αντιπροσωπεύει το sandbox περιβάλλον ανάπτυξης όπου οι developers-agents εργάζονται. Ο φάκελος περιέχει τη βασική δομή ενός παραγωγικού Flutter project, με διαχωρισμό σε πυρήνα (core), επιμέρους λειτουργίες (features), βοηθητικά εργαλεία (utils) και δοκιμές (test).

Σε κάθε ενότητα επισυνάπτεται και αντιπροσωπευτικό απόσπασμα αρχείου για την περαιτέρω επικύρωση της ποιότητας του παραγόμενου κώδικα.

```
lib
├── core
│   ├── data
│   │   └── purnari_repository.dart
│   ├── models
│   │   ├── crew_member.dart
│   │   ├── dish.dart
│   │   └── wine.dart
│   ├── navigation
│   │   ├── hash_router.dart
│   │   ├── scroll_coordinator.dart
│   │   └── sections.dart
│   ├── seo
│   │   └── meta_manager.dart
│   └── theme
│       └── app_theme.dart
├── features
│   ├── home
│   │   ├── purnari_home_page.dart
│   │   └── widgets
│   │       └── top_nav_bar.dart
│   └── sections
│       ├── about_section.dart
│       ├── contact_section.dart
│       ├── crew_section.dart
│       ├── dishes_section.dart
│       ├── hero_section.dart
│       └── wines_section.dart
├── main.dart
└── utils
    └── feedback_button.dart

11 directories, 19 files
```

Απόσπασμα 1: pubspec.yaml (dependencies & assets)

```yaml
name: xapp
description: "Awesome new Flutter AI Agent created xapp"
publish_to: 'none'
version: 1.0.0+1

environment:
  sdk: ^3.8.1

```

```
dependencies:
  flutter:
    sdk: flutter
  feedback: ^3.2.0
  http: ^1.5.0
  file_picker: ^10.3.3

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^6.0.0

flutter:
  uses-material-design: true
  assets:
    - assets/images/
    - assets/data/
```

7.1 Δομή και λογική αρχείων

Αυτή η δομή δεν είναι τυχαία: αποτελεί **τυποποιημένο pattern modular Flutter εφαρμογής** που υιοθετείται από τους agents του Appify.
Ο διαχωρισμός μεταξύ **core, features και utils** εξυπηρετεί την ανεξαρτησία των modules, επιτρέποντας στους πράκτορες να επαναχρησιμοποιούν και να αντικαθιστούν τμήματα χωρίς παρενέργειες.

```dart
import 'package:flutter/material.dart';
import 'core/theme/app_theme.dart';
import 'utils/feedback_button.dart';
import 'features/home/purnari_home_page.dart';

void main() {
  runApp(const MyApp());
}

final GlobalKey<NavigatorState> navigatorKey = GlobalKey<NavigatorState>();

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return FeedbackWrapper(
      child: MaterialApp(
        navigatorKey: navigatorKey,
        debugShowCheckedModeBanner: false,
        title: 'Purnari Tavern',
        theme: AppTheme.lightTheme,
        builder: (context, child) => Stack(children: [child!, FeedbackButton(navigatorKey: navigatorKey)]),
        home: const PurnariHomePage(),
      ),
    );
  }
}
```

7.2 Core Layer: Data, Models και Navigation

Το υποσύστημα core/ συγκεντρώνει τα θεμέλια της εφαρμογής:

core/data/purnari_repository.dart: λειτουργεί ως repository pattern για φόρτωση δεδομένων από αρχεία JSON (crew, dishes, wines). Διατηρεί caching στη μνήμη ώστε οι επαναλαμβανόμενες κλήσεις να μην απαιτούν νέα ανάγνωση.

core/models/: τρία αρχεία ορίζουν τα data models (CrewMember, Dish, Wine) με fromJson/toJson μεθόδους. Οι agents παράγουν αυτόματα αυτά τα μοντέλα, βασισμένοι στις πληροφορίες της ιδέας και της περιγραφής κάθε entity.

core/navigation/: το navigation δεν βασίζεται σε routes αλλά σε scroll-based anchors.

- Το sections.dart δηλώνει τα canonical sections (home, about, dishes, wines, crew, contact).
- Το scroll_coordinator.dart διαχειρίζεται το scrolling και ανιχνεύει την ενεργή ενότητα.
- Το hash_router.dart ενημερώνει το URL hash κατά την κύλιση, επιτρέποντας deep linking (π.χ. /#dishes).

core/seo/meta_manager.dart: το εργαλείο SEO agents προσθέτει μεταδεδομένα στο <head> (τίτλος, περιγραφή, og:image).

core/theme/app_theme.dart: ορίζει τη θεματοποίηση με σταθερές (kMaxContentWidth, kGutter) και color scheme. Το στυλ ακολουθεί Material 3 και ελαχιστοποιεί την πολυπλοκότητα για mobile/web consistency.

Τα αποσπάσματα για τα **4 πλήρη αρχεία** που αποδεικνύουν το repository pattern και τα μοντέλα:

- lib/core/data/purnari_repository.dart (ολόκληρο)

```dart
import 'dart:convert';
import 'package:flutter/services.dart';
import '../models/crew_member.dart';
import '../models/dish.dart';
import '../models/wine.dart';

class PurnariRepository {
  List<CrewMember>? _crewCache;
  List<Dish>? _dishesCache;
  List<Wine>? _winesCache;

  Future<List<CrewMember>> loadCrew() async {
    if (_crewCache != null) return _crewCache!;
    final data = await rootBundle.loadString('assets/data/crew.json');
    final List<dynamic> jsonList = json.decode(data);
    _crewCache = jsonList.map((e) => CrewMember.fromJson(e)).toList();
    return _crewCache!;
  }

  Future<List<Dish>> loadDishes() async {
    if (_dishesCache != null) return _dishesCache!;
    final data = await rootBundle.loadString('assets/data/dishes.json');
    final List<dynamic> jsonList = json.decode(data);
    _dishesCache = jsonList.map((e) => Dish.fromJson(e)).toList();
    return _dishesCache!;
  }

  Future<List<Wine>> loadWines() async {
    if (_winesCache != null) return _winesCache!;
```

```
    final data = await rootBundle.loadString('assets/data/wines.json');
    final List<dynamic> jsonList = json.decode(data);
    _winesCache = jsonList.map((e) => Wine.fromJson(e)).toList();
    return _winesCache!;
  }
}

```

- lib/core/models/crew_member.dart (ολόκληρο)

```
// CrewMember model with fromJson/toJson
class CrewMember {
  final String name;
  final String role;
  final String bio;
  final String imageAsset;
  final bool isManager;

  CrewMember({
    required this.name,
    required this.role,
    required this.bio,
    required this.imageAsset,
    required this.isManager,
  });

  factory CrewMember.fromJson(Map<String, dynamic> json) => CrewMember(
        name: json['name'] as String,
        role: json['role'] as String,
        bio: json['bio'] as String,
        imageAsset: json['imageAsset'] as String,
        isManager: json['isManager'] as bool,
      );

  Map<String, dynamic> toJson() => {
        'name': name,
        'role': role,
        'bio': bio,
        'imageAsset': imageAsset,
        'isManager': isManager,
      };
}

```

- lib/core/models/dish.dart (ολόκληρο)

```
// Dish model with fromJson/toJson
class Dish {
  final String name;
  final String description;
  final String imageAsset;

  Dish({
    required this.name,
    required this.description,
    required this.imageAsset,
  });

  factory Dish.fromJson(Map<String, dynamic> json) => Dish(
        name: json['name'] as String,
        description: json['description'] as String,
        imageAsset: json['imageAsset'] as String,
      );

  Map<String, dynamic> toJson() => {
        'name': name,
        'description': description,
        'imageAsset': imageAsset,
      };
}
```

- lib/core/models/wine.dart (ολόκληρο)

```dart
// Wine model with fromJson/toJson
class Wine {
  final String name;
  final String description;
  final String imageAsset;

  Wine({
    required this.name,
    required this.description,
    required this.imageAsset,
  });

  factory Wine.fromJson(Map<String, dynamic> json) => Wine(
        name: json['name'] as String,
        description: json['description'] as String,
        imageAsset: json['imageAsset'] as String,
      );

  Map<String, dynamic> toJson() => {
        'name': name,
        'description': description,
        'imageAsset': imageAsset,
      };
}
```

Έτσι ο αναγνώστης βλέπει την JSON-φόρτωση, caching, και τα fromJson/toJson.

7.3 Feature Layer: Οθόνες και Ενότητες

Η λογική κάθε σελίδας βρίσκεται στο φάκελο features/, με κεντρική είσοδο το home/purnari_home_page.dart.

Εδώ, οι agents δημιούργησαν ένα **stateful widget** που:

- φορτώνει τα δεδομένα από το PurnariRepository,
- παρακολουθεί scroll state μέσω ScrollCoordinator και HashRouter,
- συναρμολογεί το layout χρησιμοποιώντας τη μέθοδο _sectionWrap, που εφαρμόζει ομοιόμορφο width και padding σε κάθε ενότητα,
- αποδίδει τις υποενότητες σε δομημένη σειρά (Hero -> About-> Dishes -> Wines -> Crew -> Contact).

Κάθε ενότητα (π.χ. dishes_section.dart, crew_section.dart) είναι αυτόνομο **StatelessWidget** με σαφή ευθύνη:

- Εμφανίζει μια επικεφαλίδα (SelectableText με headline style),
- Ελέγχει αν υπάρχουν δεδομένα και αν όχι εμφανίζει fallback μήνυμα,
- Δημιουργεί δυναμικά cards (πιάτα, μέλη, κρασιά) μέσω Wrap layout που προσαρμόζεται σε mobile/desktop.

Αυτή η modular προσέγγιση ενισχύει την επεκτασιμότητα· μελλοντικές ενότητες (π.χ. Gallery, Events) μπορούν να προστεθούν χωρίς αλλαγή στην πλοήγηση

Απόσπασμα του lib/features/sections/dishes_section.dart

διότι είναι το πιο αντιπροσωπευτικό παράδειγμα modular ενότητας με δυναμική απόδοση λίστας, layout responsive, και στοιχεία κοινά με τις υπόλοιπες (wines_section.dart, crew_section.dart κ.λπ.).

```dart
import 'package:flutter/material.dart';
import '../../core/models/dish.dart';

class DishesSection extends StatelessWidget {
  final List<Dish> dishes;
  const DishesSection({super.key, required this.dishes});

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 48, horizontal: 16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          SelectableText(
            'Signature Dishes',
            style: theme.textTheme.headlineMedium,
          ),
          const SizedBox(height: 24),
          if (dishes.isEmpty)
            const Text('No dishes available at this time.')
          else
            LayoutBuilder(
              builder: (context, constraints) {
                final isMobile = constraints.maxWidth < 600;
                return Wrap(
                  spacing: 24,
                  runSpacing: 24,
                  children: dishes.map((dish) => _DishCard(dish: dish, isMobile:
isMobile)).toList(),
                );
              },
            ),
        ],
      ),
    );
  }
}

class _DishCard extends StatelessWidget {
  final Dish dish;
  final bool isMobile;
  const _DishCard({required this.dish, required this.isMobile});

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return SizedBox(
      width: isMobile ? double.infinity : 260,
      child: Card(
        elevation: 2,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
        child: Padding(
          padding: const EdgeInsets.all(16),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              ClipRRect(
                borderRadius: BorderRadius.circular(8),
                child: Image.asset(
                  dish.imageAsset,
                  height: 120,
```

```
                    width: double.infinity,
                    fit: BoxFit.cover,
                    filterQuality: FilterQuality.medium,
                    semanticLabel: dish.name,
                  ),
                ),
                const SizedBox(height: 12),
                Text(
                  dish.name,
                  style: theme.textTheme.titleMedium,
                ),
                const SizedBox(height: 6),
                Text(
                  dish.description,
                  style: theme.textTheme.bodyMedium,
                ),
              ],
            ),
          ),
        ),
      );
    }
}

```

Οι υπόλοιπες ενότητες (wines_section.dart, crew_section.dart, about_section.dart, contact_section.dart, hero_section.dart) ακολουθούν την ίδια αρχιτεκτονική λογική, Stateless Widgets που λαμβάνουν δεδομένα μέσω παραμέτρων, διαχειρίζονται τη διάταξη με responsive Wrap ή Row/Column layouts, και εφαρμόζουν κοινό στυλ από το AppTheme.

7.4 Navigation και SEO Integration

Η κλάση PurnariHomePage περιέχει το εξής μοτίβο ροής:

```
@override
void initState() {
  for (final entry in _sectionKeys.entries) {
    _scrollCoordinator.registerKey(entry.key, entry.value);
  }
  _hashRouter = HashRouter(_scrollCoordinator);
  _hashRouter.init();
  _loadData();
  MetaManager.setBaseMeta(
    title: 'Purnari Tavern',
    description: 'Where Mountain Meets the Sea...',
    imageUrl: 'assets/images/hero/hero.webp',
  );
}
```

Αυτό δείχνει την **αλληλεπίδραση των πρακτόρων**: ο Architect agent καθόρισε τις βασικές λειτουργίες (φόρτωση δεδομένων, SEO meta, pre-cache εικόνων), ενώ ο Developer agent παρήγαγε καθαρό, αναγνώσιμο Dart κώδικα σύμφωνα με best practices (asynchronous initialization, error handling, lint-clean).

Το scroll_coordinator.dart σε συνδυασμό με hash_router.dart αποδεικνύει την αυτονομία σχεδιασμού των agents — το navigation δεν είναι απλώς scroll animation, αλλά διπλής κατεύθυνσης συγχρονισμός (scroll <-> URL hash).

Απόσμασμα του lib/core/navigation/hash_router.dart,
διότι αποτυπώνει καθαρά τον μηχανισμό σύνδεσης scroll με hash-based routing και real-time ενημέρωση URL — πιο ενδεικτικό από τα υπόλοιπα navigation αρχεία.

```dart
import 'dart:async';
import 'package:flutter/foundation.dart';
import 'package:flutter/widgets.dart';
import 'sections.dart';
import 'scroll_coordinator.dart';
// ignore: avoid_web_libraries_in_flutter
import 'dart:html' as html;

class HashRouter {
  final ScrollCoordinator coordinator;
  bool _ignoreHashChange = false;
  StreamSubscription<AppSection>? _sectionSub;

  HashRouter(this.coordinator);

  void init() {
    if (!kIsWeb) return;
    WidgetsBinding.instance.addPostFrameCallback((_) {
      final hash = html.window.location.hash;
      AppSection? section;
      for (final entry in kSectionFragments.entries) {
        if (entry.value == hash) {
          section = entry.key;
          break;
        }
      }
      if (section != null) {
        coordinator.scrollTo(section, duration: Duration(milliseconds: 0));
      }
    });
    html.window.onHashChange.listen((event) {
      if (_ignoreHashChange) return;
      final hash = html.window.location.hash;
      AppSection? section;
      for (final entry in kSectionFragments.entries) {
        if (entry.value == hash) {
          section = entry.key;
          break;
        }
      }
      if (section != null) {
        coordinator.scrollTo(section);
      }
    });
    _sectionSub = coordinator.activeSection$.listen((section) {
      final fragment = kSectionFragments[section];
      if (fragment == null) return;
      if (html.window.location.hash != fragment) {
        _ignoreHashChange = true;
        html.window.history.replaceState(null, '', fragment);
        Future.delayed(const Duration(milliseconds: 100), () {
          _ignoreHashChange = false;
        });
      }
    });
  }

  void dispose() {
```

```
    _sectionSub?.cancel();
  }
}
```

Το παραπάνω αρχείο είναι χαρακτηριστικό της σχεδίασης που ακολουθούν οι agents του Appify στα συστήματα πλοήγησης: χρήση event-driven ροών (StreamSubscription), διαχείριση hash χωρίς page reload, και συντονισμός με τον ScrollCoordinator. Τα υπόλοιπα αρχεία του φακέλου navigation (scroll_coordinator.dart, sections.dart, meta_manager.dart) υλοποιούν υποστηρικτικές λειτουργίες με αντίστοιχη αρχιτεκτονική σαφήνεια.

7.5 Feedback Utility και Επεκτασιμότητα

Το αρχείο utils/feedback_button.dart αντιπροσωπεύει τη «γέφυρα» προς τους developer-agents.

Είναι ένα floating widget που:

- επιτρέπει στον τελικό χρήστη να παγώσει το UI, σημειώσει πρόβλημα ή πρόταση και να στείλει εικόνα/κείμενο στον server (/process_feedback endpoint),
- αποτυπώνει πλήρως τη φιλοσοφία **human-in-the-loop** του Appify, συνδέοντας τον τελικό χρήστη με τους πράκτορες βελτίωσης.

7.6 Αρχείο εισόδου και βασική εφαρμογή

Το main.dart θέτει το θεμέλιο της εφαρμογής:

- Τυλίγει το MaterialApp με το custom wrapper μας που δίνει την feedback λειτουργικότητα στην αναπτυσσόμενη εφαρμογή και να είναι πάντα ενεργή.
- Ορίζει το θέμα (AppTheme.lightTheme), απενεργοποιεί debug banner και χρησιμοποιεί navigatorKey για καθολικό έλεγχο των διαλόγων.
- Ορίζει ως αρχική οθόνη το PurnariHomePage.

Η απλότητα του entry point είναι σκόπιμη — οι agents ακολουθούν τον κανόνα **“keep main pure, orchestrate elsewhere**”, ώστε ο κορμός να παραμένει ανεξάρτητος από μελλοντικές αλλαγές.

Απόσπαμα του theme.dart

```
import 'package:flutter/material.dart';

class AppTheme {
  static ThemeData get lightTheme => ThemeData(
        useMaterial3: true,
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.teal),
        textTheme: const TextTheme(
          displayLarge: TextStyle(fontSize: 48, fontWeight: FontWeight.bold,
letterSpacing: -1.5),
          displayMedium: TextStyle(fontSize: 36, fontWeight: FontWeight.bold),
          displaySmall: TextStyle(fontSize: 28, fontWeight: FontWeight.bold),
          headlineMedium: TextStyle(fontSize: 24, fontWeight: FontWeight.w600),
```

```
          headlineSmall: TextStyle(fontSize: 20, fontWeight: FontWeight.w600),
          titleLarge: TextStyle(fontSize: 18, fontWeight: FontWeight.w500),
          bodyLarge: TextStyle(fontSize: 16),
          bodyMedium: TextStyle(fontSize: 14),
          bodySmall: TextStyle(fontSize: 12),
        ),
      );

  static const double kMaxContentWidth = 1200;
  static const double kGutter = 16;
}

```

7.7 Δοκιμές και Ποιοτικός Έλεγχος

Τα tests που παρήχθησαν στο φάκελο test/ αντικατοπτρίζουν τη test-driven φιλοσοφία του συστήματος:

- models_parsing_test.dart επαληθεύει round-trip fromJson/toJson για όλα τα data models.

```
// Dish model with fromJson/toJson
class Dish {
  final String name;
  final String description;
  final String imageAsset;

  Dish({
    required this.name,
    required this.description,
    required this.imageAsset,
  });

  factory Dish.fromJson(Map<String, dynamic> json) => Dish(
        name: json['name'] as String,
        description: json['description'] as String,
        imageAsset: json['imageAsset'] as String,
      );

  Map<String, dynamic> toJson() => {
        'name': name,
        'description': description,
        'imageAsset': imageAsset,
      };
}

```

- purnari_repository_test.dart ελέγχει ότι τα δεδομένα φορτώνονται επιτυχώς και δεν επιστρέφουν κενές λίστες.

```
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter/widgets.dart';
import 'package:xapp/core/data/purnari_repository.dart';

void main() {
  TestWidgetsFlutterBinding.ensureInitialized();
  final repo = PurnariRepository();

  test('loadCrew returns non-empty list', () async {
    final crew = await repo.loadCrew();
    expect(crew, isNotEmpty);
  });

  test('loadDishes returns non-empty list', () async {
    final dishes = await repo.loadDishes();
    expect(dishes, isNotEmpty);
  });

```

```
19.   test('loadWines returns non-empty list', () async {
20.     final wines = await repo.loadWines();
21.     expect(wines, isNotEmpty);
22.   });
23. }
24.
```

Οι πράκτορες δημιουργούν αυτάματα τέτοια tests με **unit granularity**, προσφέροντας εμπιστοσύνη ότι τα παραγόμενα components λειτουργούν πριν από τη μετάβαση σε integration testing.

7.8 Συνολική αποτίμηση αρχιτεκτονικής

Η ανάλυση του κώδικα δείχνει ότι το Appify:

- **αναπαράγει τυποποιημένες αρχιτεκτονικές** Flutter,
- **διαχωρίζει σαφώς τα layers** (data, navigation, UI, utilities),
- **παράγει κώδικα έτοιμο για προεπισκόπηση**, χωρίς περαιτέρω compile-time σφάλματα,
- και **εφαρμόζει best practices** (state isolation, SEO meta, modular widgets, lint consistency).

Το αποτέλεσμα είναι ένα πλήρες, συνεκτικό project, όχι πρόχειρο scaffold, που θα μπορούσε να λειτουργήσει ως θεμέλιο για πραγματικό MVP προϊόν.

4.6.3. Ενδιάμεση Συλλογιστική των Agents (τι «σκέφτονται» σε κάθε στάδιο)

Παρότι ο χρήστης βλέπει μόνο τα αποτελέσματα, κάθε πρακτορική ενέργεια διατρέχει το πρότυπο **REACT**:

Thought (Reasoning):

- **Validation:** «Η ιδέα είναι συμβατή με MVP; Αν ναι, γιατί;»
- **Brainstorming:** «Πώς αποστάζω σε 4–5 ενότητες χωρίς υπερσχεδίαση;»
- **Feature Analysis:** «Ποια πεδία είναι υποχρεωτικά; Πώς ορίζονται οι κανόνες εγκυρότητας; Πού εντάσσεται στο layout;»

Action (Τεκμηριωμένες πράξεις):

- **Επιλογή εργαλείων:** ανάγνωση τρέχοντος δέντρου lib/, επιθεώρηση μη-κενών Dart αρχείων, ενημέρωση tasks.
- **Παραγωγή/ρύθμιση artifacts:** Code Plan, λίστα widgets/οθονών, UX οδηγίες.

Observation (Μετρήσιμα αποτελέσματα):

- Έλεγχος προεπισκόπησης, συγκέντρωση warnings/lint, έλεγχος ενεργοποίησης submit, μετρικά απόκρισης UI.

Answer (Συμπέρασμα/Έξοδος):

- Τελικό σχέδιο/ενημέρωση UI, επαλήθευση acceptance criteria, αποτύπωση σε log/telemetry.

4.6.4. Σχεδιαστικές Αρχές που αποτυπώνονται στο Demo

- **MVP-first decomposition**. Κάθε ιδέα διασπάται πρώτα σε τεχνο-λειτουργικά δομικά (μοντέλα, σελίδες, widgets) και ύστερα σε ολοκλήρωση ροών (πλοήγηση, φόρμες).
- **Deterministic prompting.** Οι πράκτορες παράγουν δομημένα artifacts (λίστες, JSON-όμοιες μορφές, bullets με κριτήρια), μειώνοντας variance.
- **Context grounding.** Σε κάθε βήμα ενσωματώνεται τρέχουσα κατάσταση κώδικα (lib tree + non-empty contents), αποφεύγοντας ασάφειες.
- **Acceptance criteria per task.** Καμία εργασία δεν «κλείνει» χωρίς μετρήσιμα κριτήρια (π.χ. validation/disabled submit, lint clean).
- **HITL readiness.** Αν απαιτηθεί μελλοντικά 3rd-party integration (π.χ. αποστολή email/SMS, ημερολόγιο), η ροή προβλέπει **Human-in-the-Loop** βήματα για ασφαλή ρύθμιση.
- **Observability.** Χρονισμοί, warnings, καταγραφή αποφάσεων πρακτόρων και τελικών artifacts καταλήγουν σε τηλεμετρία για **συνεχή βελτίωση**.

4.6.5. Συνοπτική Ανάλυση Οφέλους (από το demo)

- **Χρόνος μέχρι ορατό αποτέλεσμα:** ελάχιστος. η πρώτη έκδοση είναι άμεσα live.
- **Ευελιξία:** η μετάβαση από αρχική ιδέα σε νέα λειτουργικότητα (booking form) γίνεται χωρίς ανασχεδιασμό της αρχιτεκτονικής.
- **Ποιότητα:** οι κανόνες εγκυρότητας και τα acceptance criteria εξασφαλίζουν προβλέψιμη συμπεριφορά.
- **Κλιμάκωση:** η ίδια μεθοδολογία επαναχρησιμοποιείται για επόμενα features (π.χ. gallery, menu pricing, events).

4.6.3 Μελλοντικές βελτιώσεις (engineering roadmap)

1. **Ενσωματωμένες επεξεργασίες από UI.**

Προσθήκη visual editor στην προεπισκόπηση ώστε ο τελικός χρήστης να μπορεί να αλλάζει κείμενα, padding/margins, εικόνες και βασικά στυλιστικά στοιχεία. Οι αλλαγές θα μεταφράζονται σε patches επί των σχετικών Dart widgets.

2. **Human-in-the-Loop (HITL) και 3rd-party provisioning.**

Εισαγωγή πύλης αβεβαιότητας: όταν οι πράκτορες σημειώνουν χαμηλή βεβαιότητα (π.χ. έλλειψη διαπιστευτηρίων για εξωτερικό πάροχο πληρωμών/χάρτη), το flow αναστέλλεται και ζητά στοχευμένη παρέμβαση (π.χ. εισαγωγή API keys, ρύθμιση OAuth) πριν συνεχίσει.

3. **Νέα εργαλεία πρακτόρων.**

Προσθήκη εργαλείων: TerminalUseTool (ελεγχόμενη εκτέλεση CLI με sandbox log capture), FileSearchTool (σημασιολογική και συντακτική αναζήτηση στο XAPP_DIR), FileStructureAnalysisTool (εξαγωγή δενδρικής περίληψης/μετρικών), UpdatePlanTool (αναθεώρηση σχεδίου εργασιών με βάση ευρήματα). Τα εργαλεία θα καταγράφουν λεπτομερείς οντολογικές ενέργειες για αναπαραγωγιμότητα/παρακολούθηση.

4. **“General Knowledge” αρχείο βάσης κώδικα.**

Συντήρηση ενιαίου knowledge file (π.χ. CODEBASE_KNOWLEDGE.md) με συντομευμένες συμβάσεις (naming, routing, theme), πρότυπα widgets και συνήθεις αποφάσεις αρχιτεκτονικής. Κάθε νέο αίτημα θα ξεκινάει με αυτό, μειώνοντας prompt length και variance.

5. **Ευρύτερα πειράματα LLMs/SLMs/TRM.**

Σχεδίαση benchmark ανά task-τύπο (UI synthesis, refactoring, error fixing, documentation), εξετάζοντας κόστος-απόδοση σε Μεγάλα Γλωσσικά Μοντέλα (LLMs), Μικρά Γλωσσικά Μοντέλα (SLM) και νεότερα TRM (Tiny Recursive Models). Η τηλεμετρία (AgentOps) θα συλλέγει latency/tokens/success rates ανά εργαλείο/μοντέλο για επιλογή “best-for-job”.

6. **Στοχευμένο UI/UX ανά target-group.**

Εμπλουτισμός οδηγιών Architect Agent με personas και design tokens (τυπογραφία/χρωματική παλέτα/επαναχρησιμοποιήσιμα components) ώστε οι εφαρμογές να ταιριάζουν στο κοινό-στόχο. Για παράδειγμα, “παραδοσιακή ταβέρνα” serif typography, θερμή παλέτα, μεγάλες εικόνες, προσβάσιμα CTA. Τεκμηρίωση στο knowledge file και έλεγχος συμμόρφωσης από Critic Agent.

7. **Ποιοτικός έλεγχος κώδικα και DevOps αυτοματοποίηση.**

- Unit tests on-the-fly: αυτόματη δημιουργία widget tests σε κρίσιμα components.
- Ασφάλεια API με RBAC/πολιτικές πρόσβασης και μυστικά μέσω vault.
- LLM caching (π.χ. prompt-response cache) για μείωση κόστους/latency.
- Προτυποποιημένα prompts ανά κατηγορία εφαρμογής (e-commerce, εστίαση, portfolio) για σταθερότητα αποτελεσμάτων.

8. **Computer Use Agents για αυτόματο GUI Testing:**

Η επόμενη γενιά πρακτόρων στο Appify δύναται να ενσωματώσει Computer-Use Agents, δηλαδή πράκτορες με ικανότητα εκτέλεσης πραγματικών αλληλεπιδράσεων σε γραφικά περιβάλλοντα χρήστη (GUI). Οι πράκτορες αυτοί, που βασίζονται σε προσεγγίσεις “click-through reasoning” και “visual grounding”, θα μπορούν να εκτελούν πραγματικές ενέργειες κλικ, πληκτρολόγησης και περιήγησης μέσα στις παραγόμενες Flutter εφαρμογές, λειτουργώντας ως αυτόνομοι testers. Με τον τρόπο αυτό καθίσταται δυνατή η προσομοίωση συμπεριφοράς χρήστη και η αυτόματη αξιολόγηση λειτουργικότητας (UI flow testing, navigation integrity, visual regression).

Η προσθήκη των computer-use πρακτόρων στο Appify θα επιτρέψει την ανάπτυξη ενός πλήρως αυτοματοποιημένου κύκλου QA (Quality Assurance), όπου κάθε νέα έκδοση εφαρμογής θα υποβάλλεται σε αυτοτελείς δοκιμές χωρίς ανθρώπινη

παρέμβαση. Η σύζευξη των agents αυτών με το υπάρχον REACT μοντέλο θα προσδώσει στο σύστημα τη δυνατότητα εκτέλεσης ενεργειών με βάση την οπτική αναπαράσταση της εφαρμογής, παρέχοντας μετρήσιμη αξιοπιστία και σημαντική μείωση σφαλμάτων σε σχέση με τις παραδοσιακές μεθόδους testing.

9. Ενοποίηση Agentic Προτύπων και Πολυ-Μοντελικών Ροών Συνεργασίας

Ως μελλοντική προέκταση, ιδιαίτερα ενδιαφέρουσα κατεύθυνση αποτελεί η ενσωμάτωση agentic προτύπων που αξιοποιούν την αποσύνθεση εργασιών και τη συνεργασία πολλαπλών πρακτόρων διαφορετικών δυνατοτήτων. Η προτεινόμενη κατεύθυνση συνδυάζει (i) τη μεθοδολογία κατασκευής LLM-based agents με APIs και αυτόνομη ανάθεση εργασιών, όπως περιγράφεται από τον Τζαχρήστα, [51] (ii) τη σύζευξη ετερογενών και πολυ-μοντελικών ροών που αυξάνουν την ανθεκτικότητα και την ακρίβεια των συστημάτων σε σύνθετες ρυθμίσεις [52] καθώς και (iii) μηχανισμούς αυτο-βελτίωσης μέσω αμοιβαίας αξιολόγησης και συντονισμού μεταξύ πρακτόρων.
Η υλοποίηση αυτής της προσέγγισης στο Appify θα επέτρεπε σε διαφορετικούς πράκτορες (π.χ. Architect, Developer, UI Designer, QA Tester) να συντονίζονται μέσω **κοινών στόχων**, ανταλλάσσοντας αποτελέσματα και **ανατροφοδότηση με διαφανή και αποκεντρωμένο τρόπο**. Ένα τέτοιο οικοσύστημα θα μπορούσε να οδηγήσει σε πράκτορες υψηλής αυτονομίας, με βελτιωμένη προσαρμοστικότητα και αυξημένη αποδοτικότητα χρήσης εργαλείων και υπολογιστικών πόρων, επιτρέποντας την κλιμάκωση της πλατφόρμας Appify σε πραγματικά παραγωγικά περιβάλλοντα.

10. Persona-Based Καθοδηγούμενη Παραγωγή και Στοίχιση Δεδομένων

Μια δεύτερη κατεύθυνση αφορά την προσωποκεντρική (persona-based) στοίχιση δεδομένων και την καθοδηγούμενη **διαδικασία εκπαίδευσης πρακτόρων**, με στόχο τη βελτίωση της γενίκευσης και της αξιοπιστίας τους σε πραγματικά σενάρια.
Σύμφωνα με την προσέγγιση των Tzachristas, Narayanan & Antoniou (2025) [51], η αξιοποίηση προφίλ-προσωπικοτήτων (personas) μπορεί να επιτρέψει στους πράκτορες να προσαρμόζουν τον τόνο, το ύφος και τη συμπεριφορά τους στο πλαίσιο του χρήστη, διατηρώντας παράλληλα υψηλό επίπεδο συνέπειας και ακρίβειας. Ενσωματώνοντας τέτοιους μηχανισμούς, το Appify θα μπορούσε να βελτιώσει την αντιληπτική εμπειρία του χρήστη (UX), επιτρέποντας στους AI πράκτορες να δημιουργούν εφαρμογές προσαρμοσμένες σε διαφορετικά target-groups, επίπεδα τεχνικής γνώσης ή επιχειρηματικά μοντέλα.
Η προσέγγιση αυτή θα συνέβαλε στη δημιουργία περισσότερο “ανθρωποκεντρικών” LLM-agents, ικανών να λαμβάνουν υπόψη προτιμήσεις, στυλ και περιορισμούς του εκάστοτε δημιουργού, ανοίγοντας τον δρόμο για πιο προσωποποιημένες ροές ανάπτυξης λογισμικού και βελτιωμένη ποιότητα παραγόμενου κώδικα.

Βιβλιογραφία

[1] Adamosoft, «Software Prototyping Explained: Why & How to Build a Prototype,» 24 6 2024. [Ηλεκτρονικό]. Available: https://adamosoft.com/blog/software-prototyping-explained/.

[2] AgentOps, "AgentOps — Core Concepts," 21 07 2025. [Online]. Available: https://docs.agentops.ai/core-concepts. [Accessed 16 10 2025].

[3] Gartner, «Enterprise Low-Code Application Platforms (LCAP) — Market Overview,» 2024. [Ηλεκτρονικό]. Available: https://www.gartner.com/reviews/market/enterprise-low-code-application-platform. [Πρόσβαση 15 10 2025].

[4] N. Azad και et al., «DevOps Critical Success Factors — A Systematic Literature Review,» 2023.

[5] Mind the Product, «Focusing on time to value over time to market: A strategy for developing successful products,» 2024. [Ηλεκτρονικό]. Available: https://www.mindtheproduct.com/focusing-on-time-to-value-over-time-to-market-a-strategy-for-developing-successful-products/.

[6] New Horizons, «Embracing the Future: Low Code/No Code in Citizen Development,» 2024. [Ηλεκτρονικό]. Available: https://www.newhorizons.com/resources/blog/low-code-no-code.

[7] A. Various, «The Impacts of Speed-to-Market on New Product Success,» 2025.

[8] J. Jiang, «A Survey on Large Language Models for Code Generation,» 2024.

[9] Hu et al., «Automated Design of Agentic Systems,» 2024. [Ηλεκτρονικό]. Available: https://arxiv.org/pdf/2408.08435.

[10] Flutter, «Hot reload,» 2025. [Ηλεκτρονικό]. Available: https://docs.flutter.dev/tools/hot-reload.

[11] CrewAI, «CrewAI Studio – build crews of AI agents,» 2025. [Ηλεκτρονικό]. Available: https://www.crewai.com/.

[12] LiteLLM, «LiteLLM - LLM Gateway & SDK,» 2025. [Ηλεκτρονικό]. Available: https://docs.litellm.ai/.

[13] Docker, «Docker Documentation - Build and Get Started,» 2025. [Ηλεκτρονικό]. Available: https://docs.docker.com/.

[14] tmux Project, «tmux - Getting Started / Manual,» 2024. [Ηλεκτρονικό]. Available: https://github.com/tmux/tmux/wiki/Getting-Started.

[15] Code With Andrea, «Code Generation with Dart & Flutter: The Ultimate Guide,» 2024. [Ηλεκτρονικό]. Available: https://codewithandrea.com/articles/dart-flutter-code-generation/.

[16] Bubble, «The 2024 State of No-Code: AI, Capabilities, and Trends,» 2024. [Ηλεκτρονικό]. Available: https://bubble.io/blog/state-of-no-code-development/.

[17] n8n, «AI Agents Explained: From Theory to Practical Deployment,» 2025. [Ηλεκτρονικό]. Available: https://blog.n8n.io/ai-agents/.

[18] S. Yao, «ReAct: Synergizing Reasoning and Acting in Language Models,» 2022.

[19] T. Beltramelli, «pix2code: Generating Code from a Graphical User Interface Screenshot,» 2017.

[20] M. Fowler, «Microservices,» 2014. [Ηλεκτρονικό]. Available: https://martinfowler.com/articles/microservices.html.

[21] Flutter, «Multi-Platform,» 2025. [Ηλεκτρονικό]. Available: https://flutter.dev/multi-platform.

[22] FastAPI Project, «FastAPI - Deployment and Tutorial,» 2025. [Ηλεκτρονικό]. Available: https://fastapi.tiangolo.com/.

[23] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» 2000.

[24] GitHub, «GitHub Actions documentation,» 2025. [Ηλεκτρονικό]. Available: https://docs.github.com/actions.

[25] OpenTelemetry, «OpenTelemetry — Overview,» 2025. [Ηλεκτρονικό]. Available: https://opentelemetry.io/docs/specs/otel/overview/.

[26] Wikimedia Foundation, "Flutter (software)," 15 09 2025. [Online]. Available: https://en.wikipedia.org/wiki/Flutter_(software). [Accessed 16 10 2025].

[27] Flutter (Google), "Flutter architectural overview," 01 09 2025. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview. [Accessed 16 10 2025].

[28] Flutter (Google), "Introduction to widgets," 02 09 2025. [Online]. Available: https://docs.flutter.dev/ui/widgets-intro. [Accessed 16 10 2025].

[29] Flutter (Google), "Project structure," 10 09 2025. [Online]. Available: https://docs.flutter.dev/tools/pubspec. [Accessed 16 10 2025].

[30] Flutter (Google), "Build and release a web app," 15 11 2024. [Online]. Available: https://docs.flutter.dev/deployment/web. [Accessed 16 10 2025].

[31] Flutter (Google), "Build and release an Android app," 20 02 2025. [Online]. Available: https://docs.flutter.dev/deployment/android. [Accessed 16 10 2025].

[32] Flutter (Google), "Build and release a macOS app," 18 01 2025. [Online]. Available: https://docs.flutter.dev/deployment/macos. [Accessed 16 10 2025].

[33] Wikimedia Foundation, "Dart (programming language)," 20 09 2025. [Online]. Available: https://en.wikipedia.org/wiki/Dart_(programming_language). [Accessed 16 10 2025].

[34] Dart, "dart compile," 15 09 2025. [Online]. Available: https://dart.dev/tools/dart-compile. [Accessed 16 10 2025].

[35] Dart, "dart2js: Dart-to-JavaScript compiler," 10 09 2025. [Online]. Available: https://dart.dev/tools/dart2js. [Accessed 16 10 2025].

[36] Flutter (Google), "Use the Performance view," 19 12 2024. [Online]. Available: https://docs.flutter.dev/tools/devtools/performance. [Accessed 16 10 2025].

[37] Flutter (Google), "Use the Flutter inspector," 28 08 2025. [Online]. Available: https://docs.flutter.dev/tools/devtools/inspector. [Accessed 16 10 2025].

[38] FastAPI, "FastAPI — Official documentation," 30 09 2025. [Online]. Available: https://fastapi.tiangolo.com/. [Accessed 16 10 2025].

[39] FastAPI, "Features — Pydantic & OpenAPI," 30 11 2018. [Online]. Available: https://fastapi.tiangolo.com/features/. [Accessed 16 10 2025].

[40] TechEmpower, "Web Framework Benchmarks," 01 09 2025. [Online]. Available: https://www.techempower.com/benchmarks/. [Accessed 16 10 2025].

[41] MobileDevOps, "mobiledevops/flutter-sdk-image," 01 07 2025. [Online]. Available: https://hub.docker.com/r/mobiledevops/flutter-sdk-image/. [Accessed 16 10 2025].

[42] Docker, "Compose file reference — services," 25 09 2025. [Online]. Available: https://docs.docker.com/reference/compose-file/services/. [Accessed 16 10 2025].

[43] man7.org, "tmux(1) — Linux manual page," 01 01 2024. [Online]. Available: https://man7.org/linux/man-pages/man1/tmux.1.html. [Accessed 16 10 2025].

[44] GitPython Developers, "GitPython Documentation," 24 07 2025. [Online]. Available: https://gitpython.readthedocs.io/. [Accessed 16 10 2025].

[45] PyData, "pandas documentation," 10 09 2025. [Online]. Available: https://pandas.pydata.org/docs/. [Accessed 16 10 2025].

[46] NumPy, "NumPy documentation," 05 09 2025. [Online]. Available: https://numpy.org/doc/stable/. [Accessed 16 10 2025].

[47] CrewAI, "CrewAI Documentation," 30 08 2025. [Online]. Available: https://docs.crewai.com/introduction. [Accessed 16 10 2025].

[48] BerriAI, "LiteLLM — Getting Started," 12 09 2025. [Online]. Available: https://docs.litellm.ai/docs/. [Accessed 16 10 2025].

[49] LangChain, "Serper — LangChain integration," 15 08 2025. [Online]. Available: https://python.langchain.com/docs/integrations/providers/serper. [Accessed 16 10 2025].

[50] I. (. Τζαχρήστας, «arxiv,» arxiv, 1 12 2024. [Ηλεκτρονικό]. Available: https://arxiv.org/abs/2412.13233. [Πρόσβαση 19 10 2025].

[51] I. Tzachristas, «Arxiv,» arxiv, 20 1 2025. [Ηλεκτρονικό]. Available: https://arxiv.org/abs/2501.13955. [Πρόσβαση 19 10 2025].

[52] I. T. A. S. Dominik M Weber, «Perses: Unlocking privilege escalation for small llms via extensible heterogeneity,» *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security,* pp. 344-357, 25 08 2025.