ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

ΠΜΣ «Πληροφοριακά Συστήματα και Υπηρεσίες»
Ειδίκευση «Προηγμένα Πληροφοριακά Συστήματα»
Ακαδημαϊκό έτος 2025-2026

Μάθημα: Νεφοϋπολογιστική και Προηγμένες Αρχιτεκτονικές Εφαρμογών

Θέμα: Δημιουργία Cloud-Native Web Εφαρμογής για Αλληλεπίδραση με Χαρακτήρες Rick and Morty μέσω AI και Microservices

Φοιτητής: Γεράσιμος Πλατής
Α.Μ.: me25081

Ημερομηνία: Ιανουάριος-Φεβρουάριος 2026

ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ

Κεφάλαιο 1: Εισαγωγή και περιγραφή του σεναρίου

Η παρούσα εργασία αφορά την ανάπτυξη μιας cloud-native εφαρμογής βασισμένης σε αρχιτεκτονική microservices. Το επιλεγμένο σενάριο υλοποιεί έναν δυναμικό ευρετήριο χαρακτήρων (Character Explorer), ο οποίος αξιοποιεί ζωντανά δεδομένα από το εξωτερικό Rick and Morty API μέσω RESTful διεπαφών.

Η εφαρμογή επιτρέπει στον χρήστη να αλληλεπιδρά με πληροφορίες χαρακτήρων σε πραγματικό χρόνο και να δημιουργεί και να διαχειρίζεται μία τοπική συλλογή αγαπημένων χαρακτήρων (favourite collection), η οποία αποθηκεύεται σε βάση δεδομένων CouchDB. Παράλληλα, ο χρήστης έχει τη δυνατότητα να λαμβάνει εμπλουτισμένες πληροφορίες σε μορφή fun facts και σεναρίων «διάδρασης» μεταξύ βασικών χαρακτήρων της σειράς και των χαρακτήρων που έχει ορίσει ως αγαπημένους, οι οποίες παράγονται μέσω Τεχνητής Νοημοσύνης (Generative AI) με αξιοποίηση του Groq API.

Η προτεινόμενη λύση ακολουθεί τις αρχές του cloud-native σχεδιασμού, επιτυγχάνοντας υψηλή επεκτασιμότητα (scalability), αποσύζευξη (decoupling) και ευελιξία, καθώς κάθε λειτουργικότητα υλοποιείται ως ανεξάρτητο microservice.

Κεφάλαιο 2: Περιγραφή της αρχιτεκτονικής συστήματος

2.1 Μοντέλο Microservices

Στην εφαρμογή αυτή δεν ακολουθήθηκε η παραδοσιακή «μονολιθική» προσέγγιση, αλλά το πρότυπο των **Microservices**. Ουσιαστικά, η εφαρμογή "έσπασε" σε μικρότερα, αυτόνομα κομμάτια που το καθένα εκτελεί μια συγκεκριμένη λειτουργία μέσα στο δικό του **Container**. Έτσι, οι ρυθμίσεις της μίας υπηρεσίας δεν μπερδεύονται με της άλλης.

Η επιλογή αυτή έγινε για τρεις βασικούς λόγους:

- **Αποσύζευξη (Decoupling):** Το UI, οι βάσεις δεδομένων και τα API λειτουργούν ανεξάρτητα. Αν, για παράδειγμα, χρειαστεί να κάνουμε μια αλλαγή στην CouchDB, δεν χρειάζεται να πειράξουμε καθόλου τους JSON-Servers.

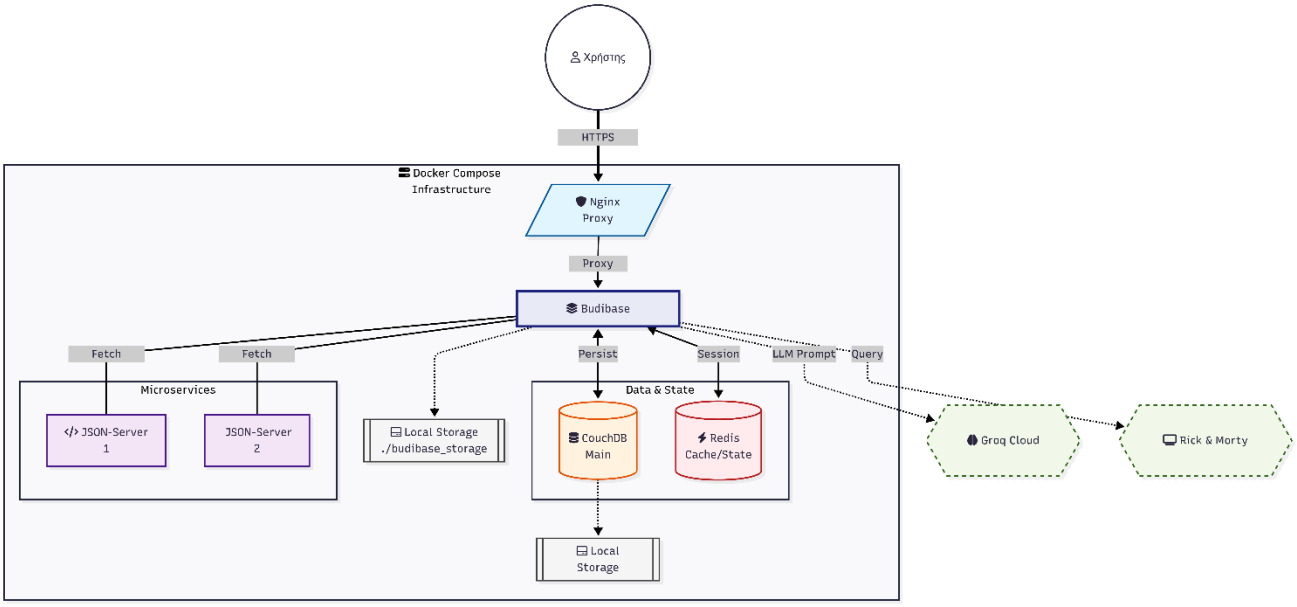
- **Ευελιξία και Επεκτασιμότητα:** Αν στο μέλλον η κίνηση αυξηθεί, μπορούμε να ενισχύσουμε μόνο το κομμάτι που πιέζεται (π.χ. τον Nginx) χωρίς να δεσμεύουμε πόρους για όλο το σύστημα.

- **Ανθεκτικότητα:** Αν ένα container κολλήσει (π.χ. το Portainer), η υπόλοιπη εφαρμογή συνεχίζει να τρέχει κανονικά, διασφαλίζοντας ότι ο χρήστης δεν θα βρει μπροστά του ένα ολικό σφάλμα.

Όσον αφορά την επικοινωνία, όλα τα containers είναι συνδεδεμένα σε ένα κοινό εσωτερικό δίκτυο (**Docker Bridge Network**). Αυτό μας λύνει τα χέρια, καθώς οι υπηρεσίες "βλέπουν" η μία την άλλη χρησιμοποιώντας απλά το όνομά τους (π.χ. http://jsonserver:3000) και όχι σταθερές διευθύνσεις IP που μπορεί να αλλάξουν.

2.2 Τεχνολογίες και Επιλογές

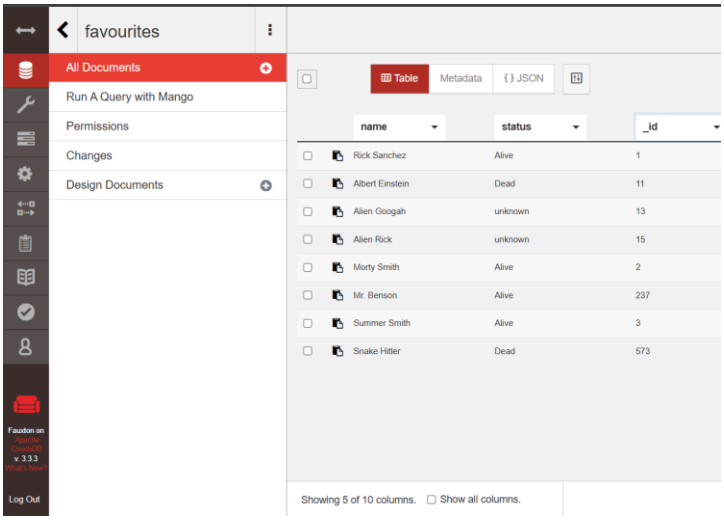
Για να καταλάβουμε πώς λειτουργεί η εφαρμογή στην πράξη, το παρακάτω διάγραμμα δείχνει πώς "μιλούν" τα containers μεταξύ τους: ο Nginx υποδέχεται τον χρήστη, το Budibase αναλαμβάνει τον συντονισμό, ενώ οι βάσεις δεδομένων και η τεχνητή νοημοσύνη (Groq) δουλεύουν στο παρασκήνιο για να προσφέρουν το τελικό αποτέλεσμα.



Nginx (Gateway Layer):. Η επιλογή του Nginx έγινε με γνώμονα τη δυνατότητα έκθεσης της εφαρμογής εκτός τοπικού δικτύου (Port Forwarding). Ο Nginx λειτουργεί ως το μοναδικό σημείο επαφής με το εξωτερικό δίκτυο, προστατεύοντας τις εσωτερικές θύρες των microservices

Budibase (UI & Orchestration): Η κύρια πλατφόρμα ανάπτυξης του περιβάλλοντος εργασίας. Ενορχηστρώνει τη ροή δεδομένων, συνδέοντας τις βάσεις δεδομένων και τα REST APIs, ενώ διαχειρίζεται τα αιτήματα προς το **Groq LLM** για τη δημιουργία δυναμικού περιεχομένου.

CouchDB (Data Layer): επιλέχθηκε επειδή είναι document-oriented, κάτι που ταιριάζει απόλυτα με το JSON format του Rick and Morty API. Ένα πλεονέκτημα της επιλογής της CouchDB είναι το ενσωματωμένο περιβάλλον διαχείρισης Fauxton (http://localhost:5984/_utils/#login).Πρόκειται για ένα Web GUI που επιτρέπει την άμεση εποπτεία της βάσης, τη διαχείριση των JSON εγγράφων και την εκτέλεση queries χωρίς την ανάγκη εξωτερικών εργαλείων. Η χρήση του Fauxton ήταν καθοριστική κατά τη φάση της ανάπτυξης για την επαλήθευση της σωστής αποθήκευσης των "αγαπημένων" χαρακτήρων και τον έλεγχο της δομής των δεδομένων που επιστρέφονταν από τα API transformations.

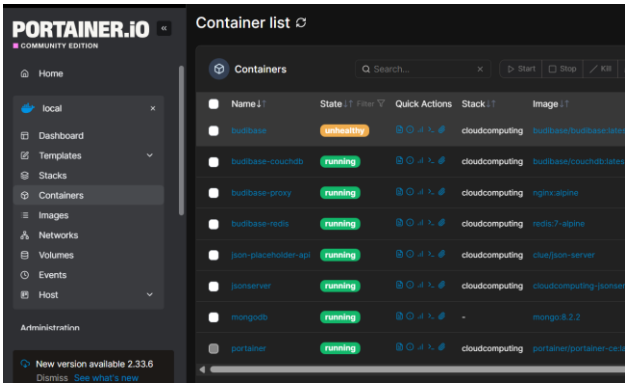


Redis (State Management): Υπηρεσία in-memory αποθήκευσης βασισμένη στο Redis 7-Alpine image, επιλεγμένη για το ελάχιστο αποτύπωμα μνήμης και την ταχύτητά της. Πέρα από τη διαχείριση των sessions, η Redis λειτουργεί ως αυτόνομο microservice για την καταγραφή ενεργειών (Audit Logging). Χρησιμοποιώντας τη δομή δεδομένων Redis Lists, η εφαρμογή αποθηκεύει δυναμικά ιστορικό κλήσεων (π.χ. πότε ανοίχτηκε ένας χαρακτήρας ή πότε έγινε κλήση στο AI), επιτρέποντας στον διαχειριστή να παρακολουθεί τη ροή δεδομένων σε πραγματικό χρόνο μέσω του Redis CLI.

JSON-Server (Context API): Microservice που παρέχει τα δεδομένα της "οικογένειας" Rick and Morty. Λειτουργεί ως η βασική πηγή context για το **Groq AI**, επιτρέποντας τη δημιουργία ρεαλιστικών διαλόγων και σεναρίων διάδρασης.

JSON-Placeholder (Frontend API): Ανεξάρτητο mock API που τροφοδοτεί αποκλειστικά την αρχική σελίδα της εφαρμογής.

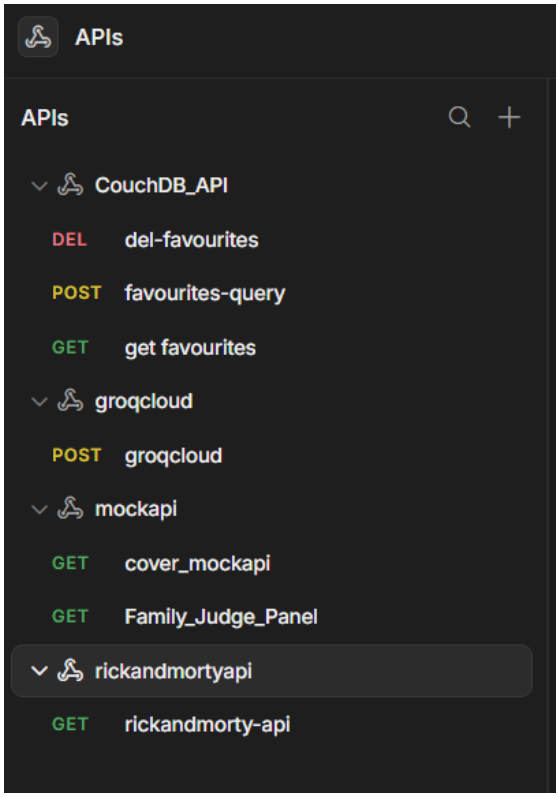
Portainer (Infrastructure Management): Εργαλείο εποπτείας της Docker υποδομής. Χρησιμοποιείται για τη διαχείριση των containers, την παρακολούθηση των logs και τον έλεγχο των πόρων του συστήματος σε πραγματικό χρόνο.



Κεφάλαιο 3: Περιγραφή Λειτουργικών Ροών (API Workflows)

Όπως φαίνεται στη δομή της εφαρμογής, έχουν υλοποιηθεί τέσσερις κύριοι άξονες διασύνδεσης:

1. **Διαχείριση Προτιμήσεων (CouchDB API)**
   Η ροή αυτή αφορά την αλληλεπίδραση του χρήστη με τη βάση δεδομένων για την αποθήκευση των επιλογών του:
   - **get favourites (GET):** Ανάκτηση της λίστας με τους χαρακτήρες που έχει αποθηκεύσει ο χρήστης ως "αγαπημένους"
   - **favourites-query (POST):** Αναζήτηση και φιλτράρισμα συγκεκριμένων εγγραφών μέσα στη βάση CouchDB
   - **del-favourites (DEL):** Διαγραφή χαρακτήρων από τη λίστα προτιμήσεων, επιτρέποντας στον χρήστη να ανανεώνει το context του.
2. **Παραγωγή AI Περιεχομένου (GroqCloud API)**
   Αυτή είναι η κεντρική "έξυπνη" ροή της εφαρμογής **groqcloud (POST):** Στέλνει το τελικό prompt στο Groq LLM. Το prompt περιλαμβάνει τα δεδομένα των αγαπημένων (από CouchDB) και το context της οικογένειας (από JSON-Server). Η ροή επιστρέφει το δημιουργημένο σενάριο αλληλεπίδρασης σε πραγματικό χρόνο. Για την εξαγωγή της πληροφορίας από το **Groq API**, εφαρμόστηκε η τεχνική του **Output Parsing** μέσω JavaScript. Επειδή η απόκριση του γλωσσικού μοντέλου (LLM) είναι σε ελεύθερη μορφή κειμένου, χρησιμοποιήθηκε ένας προκαθορισμένος delimiter |||. Με τη χρήση της μεθόδου split(), η εφαρμογή διαχωρίζει δυναμικά το "Fact" από το "Verdict", επιτρέποντας την προβολή της κατάλληλης πληροφορίας στα αντίστοιχα UI components, διασφαλίζοντας έτσι τη δομημένη παρουσίαση των μη-δομημένων δεδομένων της AI
3. **Τροφοδοσία Δυναμικού Context (Mock API - JSON-Server)**
   Εδώ πραγματοποιείται η άντληση του "στατικού" context που απαιτείται για την AI.
   - **Family_Judge_Panel (GET):** Αντλεί τα δεδομένα των μελών της οικογένειας Rick and Morty από τον jsonserver. Αυτά τα δεδομένα χρησιμεύουν ως το "υπόβαθρο" (background) πάνω στο οποίο η AI θα βασίσει τους διαλόγους της.
   - **cover_mockapi (GET):** Παρέχει τα απαραίτητα δεδομένα (εικόνες ή κείμενα) για το εικαστικό μέρος της αρχικής σελίδας μέσω του jsonplaceholder.
4. **Εξωτερική Πληροφόρηση (Rick and Morty API): rickandmorty-api (GET):** Πραγματοποιεί κλήσεις



προς το επίσημο εξωτερικό API για την αναζήτηση και εύρεση νέων χαρακτήρων, οι οποίοι στη συνέχεια μπορούν να αποθηκευτούν στην CouchDB.

Σύνοψη Διασύνδεσης

Ουσιαστικά, το **Budibase** λειτουργεί ως "κόμβος":

1. **Συλλέγει** δεδομένα από το mockapi και το rickandmortyapi.
2. **Αποθηκεύει/Διαχειρίζεται** τις επιλογές στην CouchDB_API.
3. **Συνθέτει** όλα τα παραπάνω σε ένα αίτημα προς το groqcloud για να παράξει το τελικό αποτέλεσμα.

Κεφάλαιο 4: Ενορχήστρωση με Docker Compose

4.1 Σκεπτικό και Υλοποίηση με Containers

Για το στήσιμο της εφαρμογής επιλέχθηκε η λύση του Docker, καθώς επιτρέπει τον διαχωρισμό των διαφορετικών υπηρεσιών σε απομονωμένα περιβάλλοντα. Αντί να εγκαθιστούμε κάθε υπηρεσία ξεχωριστά στο λειτουργικό σύστημα, χρησιμοποιήσαμε το Docker Compose για να "σηκώσουμε" όλο το stack (βάσεις, APIs, UI) ταυτόχρονα, διασφαλίζοντας ότι όλα επικοινωνούν σωστά μεταξύ τους.

4.2 Ανάλυση των Υπηρεσιών (Services)

Στο αρχείο docker-compose.yaml ορίστηκαν συνολικά 7 containers:

- Budibase, CouchDB & Redis: Αποτελούν τον βασικό κορμό της εφαρμογής. Χρησιμοποιήθηκε η παράμετρος depends_on ώστε το Budibase να περιμένει την ετοιμότητα των βάσεων πριν ξεκινήσει.
- Mock APIs (jsonserver & json-placeholder): Δημιουργήθηκαν δύο αυτόνομες υπηρεσίες για τα δεδομένα των χαρακτήρων και της αρχικής σελίδας, επιτρέποντας την τροποποίηση των δεδομένων χωρίς να επηρεάζεται η υπόλοιπη εφαρμογή.
- Nginx Proxy: Λειτουργεί ως κεντρική πύλη εισόδου στη θύρα 80, αναλαμβάνοντας τη δρομολόγηση προς τα εσωτερικά containers και λύνοντας ζητήματα συμβατότητας με τα ports.
- Portainer: Προστέθηκε ως γραφικό περιβάλλον διαχείρισης για τον έλεγχο της κατάστασης των containers και την ανάγνωση των logs κατά το debugging.

4.3 Τοπική Αποθήκευση και Επίλυση Απώλειας Δεδομένων

Κατά τις πρώτες δοκιμές, αντιμετωπίστηκε το πρόβλημα της απώλειας δεδομένων, καθώς κάθε φορά που γινόταν επανεκκίνηση ή ανακατασκευή των containers (compose down/up), η βάση δεδομένων CouchDB επανερχόταν στην αρχική της κενή κατάσταση. Για να λυθεί αυτό, αντιστοιχίσαμε τα volumes τοπικά στον φάκελο του project μας, συνδέοντας τους φακέλους των containers με καταλόγους όπως το ./budibase_storage και το ./redis_storage. Με αυτή τη ρύθμιση, τα δεδομένα των χαρακτήρων και οι ρυθμίσεις του Budibase αποθηκεύονται πλέον μόνιμα στον φυσικό δίσκο του υπολογιστή μας. Έτσι, η βάση παραμένει ανέπαφη ακόμα και αν διαγραφούν ή αναβαθμιστούν τα containers. Παράλληλα, έχουμε άμεση πρόσβαση στα αρχεία JSON (π.χ. στον φάκελο ./mock-api) για γρήγορες αλλαγές στο context της εφαρμογής.

4.4 Δικτύωση και Επικοινωνία

Όλα τα containers εντάχθηκαν στο εσωτερικό δίκτυο budibase_network. Αυτό επέτρεψε τη χρήση Service Discovery, όπου η επικοινωνία μεταξύ των υπηρεσιών γίνεται μέσω των ονομάτων τους (π.χ. http://jsonserver:3000) αντί για IPs. Η χρήση του bridge driver απομόνωσε τα APIs από το εξωτερικό δίκτυο, αφήνοντας μόνο τον Nginx εκτεθειμένο για την πρόσβαση του χρήστη.

4.5 Παραμετροποίηση μέσω .env

Για την ευκολία των δοκιμών, όλες οι κρίσιμες ρυθμίσεις όπως οι θύρες (ports) και οι κωδικοί πρόσβασης των βάσεων τοποθετήθηκαν σε ένα αρχείο .env. Με αυτόν τον τρόπο, η εφαρμογή έγινε πιο ευέλικτη, καθώς μπορούσαμε να αλλάξουμε τις ρυθμίσεις του περιβάλλοντος χωρίς να τροποποιούμε τον κώδικα του Docker Compose.

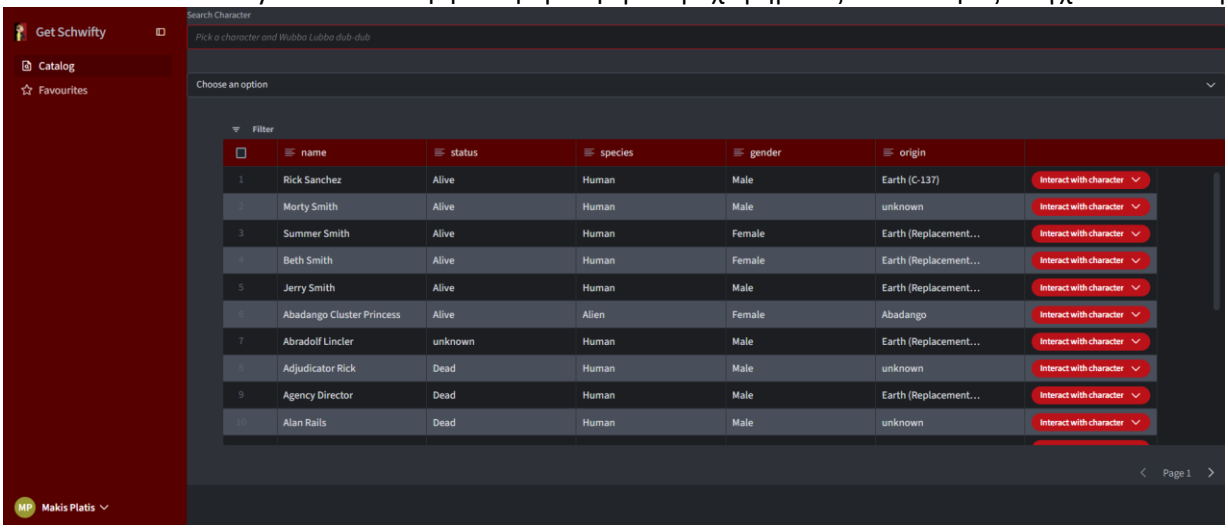
Κεφάλαιο 5: Υλοποίηση και Σενάριο Χρήσης

5.1 Σελίδα Υποδοχής (Landing Page)

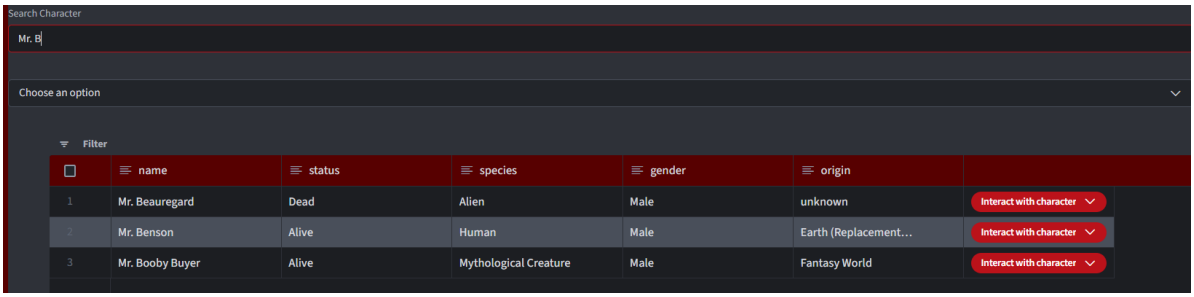
Η εφαρμογή ξεκινά με μια σελίδα υποδοχής που λειτουργεί ως η πρώτη επαφή του χρήστη με το σύστημα. Σε αυτό το στάδιο, το Budibase ανακτά δεδομένα από το JSON-Placeholder (cover_mockapi) και προβάλλει μια σειρά χαρακτήρων ως δείγμα. Η σελίδα περιλαμβάνει το κεντρικό action button "Start App", το οποίο εκτελεί τη μετάβαση (navigation) από το στατικό περιβάλλον στο κύριο ταμπλό διαχείρισης της εφαρμογής.

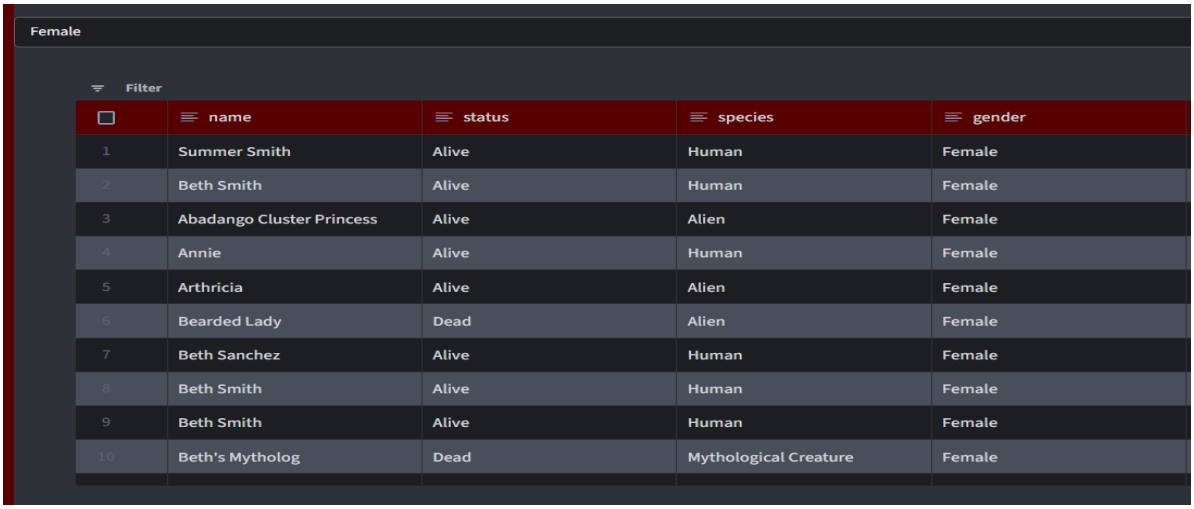
5.2 Κύριος Κατάλογος Χαρακτήρων (Data Catalog)

Μπαίνοντας στο κύριο περιβάλλον, ο χρήστης αποκτά πρόσβαση στον πλήρη κατάλογο που τροφοδοτείται από το Rick and Morty API. Η διεπαφή αυτή προσφέρει προχωρημένες δυνατότητες ελέγχου των δεδομένων:

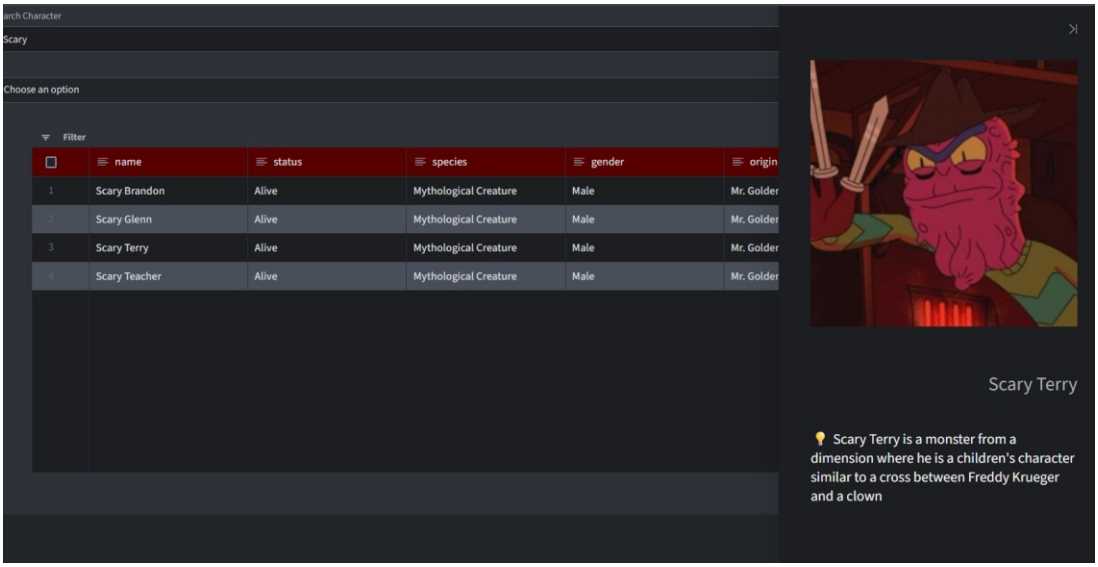


a. Αναζήτηση και Φιλτράρισμα: Ενσωματώθηκε Search Bar για εύρεση βάσει ονόματος και Gender Picker για φιλτράρισμα βάσει φύλου. Κάθε αλλαγή στα φίλτρα πυροδοτεί νέα API calls.





b. Σελιδοποίηση (Paging): Υλοποιήθηκε σύστημα σελίδων για την αποδοτική πλοήγηση στον μεγάλο όγκο δεδομένων του API.
c. Αλληλεπίδραση (Interact with Character): Για κάθε χαρακτήρα παρέχεται ένα κουμπί που ανοίγει δύο επιλογές:
d. Mark as Favourite: Πραγματοποιεί POST request στην CouchDB, αποθηκεύοντας τον χαρακτήρα στην τοπική βάση.
e. Open Character: Ανοίγει ένα Side Panel που φορτώνει τη φωτογραφία και το όνομα του χαρακτήρα, ενώ μέσω του Groq API γίνεται generate ένα "Fun Fact" σε πραγματικό χρόνο.



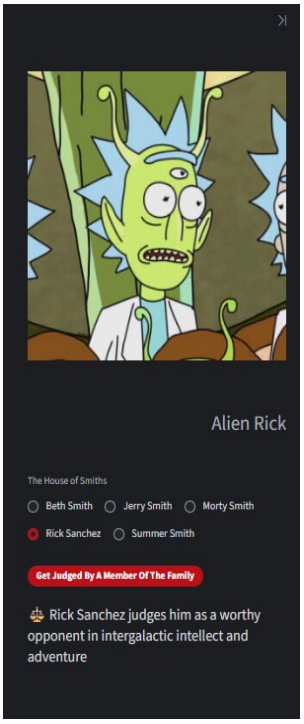
5.3 Σελίδα Αγαπημένων και AI Judging (Favourites Page)

Η δεύτερη σελίδα της εφαρμογής αφορά τη διαχείριση των αποθηκευμένων χαρακτήρων από την CouchDB. Εδώ ο χρήστης μπορεί να δει τη λίστα των αγαπημένων του και να αλληλεπιδράσει εκ νέου μαζί τους:

- Remove Character: Ο χρήστης μπορεί να αφαιρέσει έναν χαρακτήρα μέσω DELETE request, ενημερώνοντας το persistence layer της CouchDB.

- Open Character & Family Selection: Ανοίγοντας το Side Panel ενός αγαπημένου χαρακτήρα, εμφανίζεται η φωτογραφία και το όνομά του, αλλά και μια νέα λίστα με τα μέλη της βασικής οικογένειας της σειράς.

- Get Judged by a Member of the Family: Αφού ο χρήστης επιλέξει ένα μέλος της οικογένειας, ενεργοποιείται ένα κουμπί που στέλνει ένα σύνθετο prompt στο Groq API. Η AI αναλαμβάνει τον ρόλο του "δικαστή" και παράγει ένα σχόλιο για τον χαρακτήρα, βασισμένο στην προσωπικότητα του μέλους της οικογένειας που επιλέχθηκε.



5.4 Τεχνική Αποτίμηση Λειτουργίας

Η υλοποίηση του σεναρίου χρήσης απέδειξε τη σταθερότητα της αρχιτεκτονικής. Η χρήση του Budibase επέτρεψε το γρήγορο "δέσιμο" των UI στοιχείων με τις API κλήσεις, ενώ ο συνδυασμός της CouchDB για το CRUD κομμάτι και του Groq για το δυναμικό περιεχόμενο, δημιούργησε μια ολοκληρωμένη full-stack εμπειρία μέσα σε ένα containerized περιβάλλον.

Κεφάλαιο 6: Τεχνικές Προκλήσεις και Επιλύσεις

Κατά τη διάρκεια της υλοποίησης, προέκυψαν τεχνικά ζητήματα που απαίτησαν εξειδικευμένες ρυθμίσεις σε επίπεδο υποδομής και κώδικα:

6.1 Διαχείριση Persistence και Docker Lifecycle

Η αρχική προσέγγιση εμφάνισε απώλεια δεδομένων στην CouchDB μετά από κάθε επανεκκίνηση των containers, οπότε και υλοποιήθηκε η αντιστοίχιση τοπικών volumes, διασφαλίζοντας ότι τα δεδομένα παραμένουν τοπικά ανεξάρτητα από την κατάσταση του container.

6.2 Ενορχήστρωση και Δικτύωση (Orchestration)

Εξαρτήσεις (Dependencies) & Διαγνωστικός Έλεγχος: Η ταυτόχρονη εκκίνηση των υπηρεσιών προκάλεσε αρχικά σφάλματα σύνδεσης, τα οποία αντιμετωπίστηκαν με την παράμετρο depends_on. Μέσω του Portainer, εντοπίστηκε ότι το container του Budibase εμφάνιζε ένδειξη "unhealthy". Η ανάλυση των logs αποκάλυψε ότι η αιτία δεν ήταν η ίδια η εφαρμογή, αλλά η αδυναμία ταυτοποίησης με τον Redis (AUTH failed). Επιλέχθηκε η διατήρηση απλών health checks αντί για αυστηρούς περιορισμούς, ώστε το Budibase να παραμένει σε λειτουργία ακόμα και σε κατάσταση unhealthy. Αυτό επέτρεψε την απρόσκοπτη πρόσβαση στο Monitor/Logs, διευκολύνοντας τον εντοπισμό του σφάλματος στις ρυθμίσεις του Redis χωρίς να προκληθεί deadlock (πάγωμα) ή συνεχείς επανεκκινήσεις του stack κατά τη διάρκεια του development.

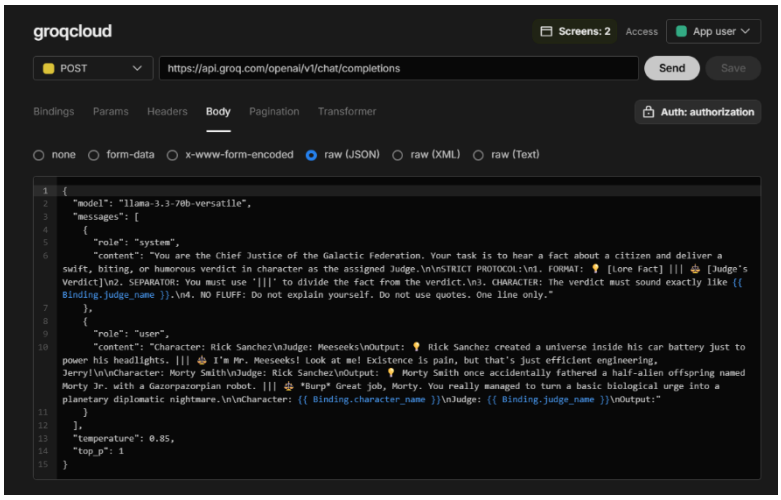
Conflict Θυρών: Υπήρξαν συγκρούσεις θυρών με άλλες υπηρεσίες του συστήματος. Το πρόβλημα λύθηκε με ανακατεύθυνση των εσωτερικών θυρών των microservices σε ελεύθερες εξωτερικές θύρες μέσω του Nginx gateway.

6.3 API Integration και Data Flow

Caching & Synchronization: Κατά τις δοκιμές των API calls, το περιβάλλον του Budibase διατηρούσε στην προσωρινή μνήμη (cache) παλαιότερα responses, εμποδίζοντας την αποθήκευση νέων αλλαγών. Η επίλυση απαιτούσε συχνή εκκαθάριση της προσωρινής μνήμης του browser (Hard Reload - Ctrl+F5) και επαλήθευση των headers.
Data Bindings & Transformations: Η διασύνδεση των δυναμικών πεδίων (bindings) μεταξύ των API responses και των UI στοιχείων παρουσίασε πολυπλοκότητα. Χρησιμοποιήθηκαν JavaScript Transformations για τον μετασχηματισμό των JSON δεδομένων σε πραγματικό χρόνο, ώστε να είναι συμβατά με τα inputs των επόμενων κλήσεων.

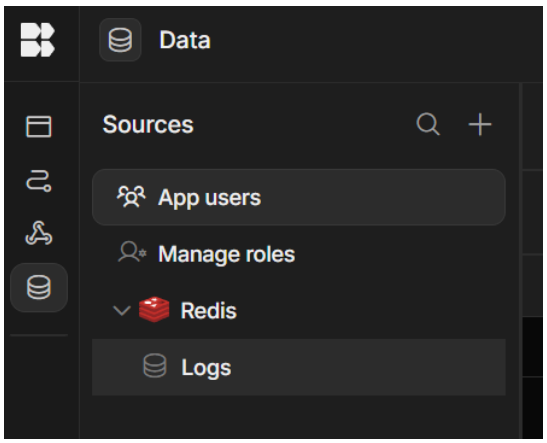
6.4 Βελτιστοποίηση Generative AI (GroqCloud)

Η παραγωγή κειμένου απαιτούσε διαδοχικές δοκιμές για την επίτευξη του επιθυμητού αποτελέσματος:
Επιλογή Μοντέλου: Έγινε μετάβαση στο μοντέλο Llama-3.3-70b-versatile του Groq, το οποίο προσφέρει την ιδανική ισορροπία μεταξύ ταχύτητας απόκρισης και ικανότητας τήρησης σύνθετων οδηγιών μορφοποίησης.
Ρύθμιση Παραμέτρων (Temperature): Η παράμετρος Temperature ρυθμίστηκε στο 0.7 - 0.85. Η τιμή αυτή επιλέχθηκε ώστε οι αποκρίσεις να έχουν τη δημιουργικότητα που απαιτεί το σενάριο, παραμένοντας όμως εντός των ορίων που διασφαλίζουν τη σωστή χρήση του διαχωριστή |||.
Δομή και Καθοδήγηση (System Prompting): Σχεδιάστηκαν αυστηρά prompts με τη χρήση παραδειγμάτων (few-shot), που υποχρεώνουν την AI να επιστρέφει την πληροφορία σε μορφή μίας γραμμής. Αυτό απέτρεψε την παραγωγή περιττών σχολίων που θα δυσκόλευαν την αυτόματη επεξεργασία της απάντησης.
Διαχείριση Σφαλμάτων μέσω Scripting: Ενσωματώθηκε κώδικας JavaScript (Optional Chaining) για την επεξεργασία της απάντησης. Αυτό εξασφαλίζει ότι η εφαρμογή μπορεί να διαχειριστεί τυχόν αστοχίες του API ή κενές αποκρίσεις, προβάλλοντας μηνύματα αναμονής (π.χ. "Waiting for data...") αντί για τεχνικά σφάλματα.



```
var data = $("[New Data Provider].Rows");

// 1. Check if the array exists AND the specific row [0] exists
if (!data || !data[0]) {
    return "Waiting for data...";
}

var choice = data[0]?.choices?.[0];

// 2. Check if the API returned an error object instead of a choice
if (data[0].error) {
    return "Groq API Error: " + data[0].error.message;
}

// 3. Extract content
var content = choice?.message?.content;

if (content) {
    return content.includes("|||") ? content.split("|||")[0].trim() : content.trim();
}

return "Response received, but no text content found.";
```

6.5 Υλοποίηση Microservice Logging και Circular Buffer στη Redis



Κατά την τελική φάση της ανάπτυξης, κρίθηκε αναγκαία η προσθήκη ενός στρώματος καταγραφής ενεργειών (logging layer) για την εποπτεία της εφαρμογής σε πραγματικό χρόνο. Αντί για την παραδοσιακή αποθήκευση σε στατικό αρχείο, υλοποιήθηκε ένα αυτόνομο microservice εντός του Redis Alpine container.

Αρχιτεκτονική Circular Buffer: Για τη βέλτιστη διαχείριση της in-memory μνήμης, χρησιμοποιήθηκε η τεχνική του κυλιόμενου buffer. Μέσω της εντολής LTRIM logs:{{user_id}} 0 99, η εφαρμογή διατηρεί αυτόματα μόνο τις 100 πιο πρόσφατες καταχωρήσεις ανά χρήστη, διαγράφοντας τις παλαιότερες.
Κάθε αλληλεπίδραση (π.χ. διαγραφή χαρακτήρα, άνοιγμα panel) εκτελεί μια ατομική δέσμη εντολών (LPUSH, LLEN, EXPIRE), εξασφαλίζοντας ότι το log συνοδεύεται από ακριβές timestamp και αυτόματη εκκαθάριση (TTL) μετά από 1200 δευτερόλεπτα αδράνειας.

Προκειμένου να διευκολυνθεί η κεντρική εποπτεία του συστήματος από τον διαχειριστή, υλοποιήθηκε η δυνατότητα ομαδοποιημένης ανάκτησης logs (bulk retrieval) μέσω Lua Scripting στο CMD δημιουργώντας με την εντολή

docker exec -it budibase-redis redis-cli

EVAL "local keys = redis.call('keys', 'logs:*') local total_logs = {} for i, k in ipairs(keys) do table.insert(total_logs, k) table.insert(total_logs, redis.call('LRANGE', k, 0, -1)) end return total_logs" 0

συνεδρία στο container που τρέχει τη Redis Alpine. Η χρήση της εντολής EVAL επιτρέπει την εκτέλεση κώδικα απευθείας στον Redis server, ο οποίος:

- **Ανακτά** δυναμικά όλα τα κλειδιά που ακολουθούν το pattern logs:*.
- **Επαναλαμβάνει** (iterate) τη διαδικασία ανάγνωσης για κάθε χρήστη.
- **Επιστρέφει** ένα ενιαίο δομημένο αποτέλεσμα με όλα τα ιστορικά δεδομένα.

6.5 Μελλοντικές Βελτιώσεις (Σε περιβάλλον production)

Διαχείριση Ευαίσθητων Δεδομένων: Στην τρέχουσα υλοποίηση, το API Key του Groq έχει εισαχθεί απευθείας στο API Connection του Budibase για λόγους ταχύτητας ανάπτυξης. Ως μελλοντική βελτίωση, προτείνεται η μεταφορά του στο αρχείο .env του Docker. Η πρόσβαση από το UI θα μπορούσε να επιτευχθεί μέσω των **Environment Variables** του Budibase Server, εξασφαλίζοντας ότι το κλειδί δεν θα είναι ορατό στο Frontend (Client-side), ενισχύοντας έτσι την ασφάλεια της εφαρμογής.
Διαχείριση και μείωση των API calls: Ως κεντρική μελλοντική αναβάθμιση προτείνεται η ενσωμάτωση μιας στρατηγικής Caching μέσω της υπηρεσίας Redis, με στόχο τη μείωση των αιτημάτων προς τα εξωτερικά APIs (Groq & Rick and Morty). Η αποθήκευση των συχνά ζητούμενων δεδομένων στην in-memory βάση θα μειώσει δραστικά το latency της εφαρμογής και θα περιορίσει την κατανάλωση API tokens.

Κεφάλαιο 7: GitHub Repository

https://github.com/makisplts1995/cloud-native-character-portal

Κεφάλαιο 8: Βιβλιογραφία

1. **Docker Documentation:** Docker Compose V2 Specification. Διαθέσιμο στο: https://docs.docker.com/compose/
2. **Apache CouchDB Documentation:** HTTP API Reference. Διαθέσιμο στο: https://docs.couchdb.org/en/stable/api/index.html
3. **Budibase Guide:** REST API Integration & Data Providers. Διαθέσιμο στο: https://docs.budibase.com/docs/rest-api
4. **Groq Cloud:** API Reference & Prompt Engineering Documentation. Διαθέσιμο στο: https://console.groq.com/docs/quickstart
5. **The Rick and Morty API:** Documentation for RESTful character data. Διαθέσιμο στο: https://rickandmortyapi.com/
6. **Portainer Documentation:** Container Management and Logs. Διαθέσιμο στο: https://docs.portainer.io/
7. Για το διάγραμμα χρησιμοποιήθηκε το εργαλείο Mermaid.js. Markdown-based diagramming and charting tool Διαθέσιμο στο: https://mermaid.js.org/