TECHNOLOGIA

ROLA I ZNACZENIE ARCHITEKTURY W OBSZARZE IT

dr hab. inż. Jan Werewka, prof. WSEI

STRESZCZENIE

Znaczenie architektury w budowaniu różnych obiektów jest ogromne. Tymczasem przy budowaniu systemów IT wiele programistów wydaje się nie przywiązywać wagi do architektury. Powstaje pytanie dlaczego tak się dzieje i czy stanowi to zagrożenie dla budowanych systemów IT. W opracowaniu postarano się na początku uporządkować pojęcia z zakresu architektury systemów IT. Rozpoczęto od ogólnych pojęć z tego zakresu i przedstawienia właściwości intuicyjnie rozumianej architektury budowlanej. Następnie dokonano przeglądu podstawowych architektur IT, a dla wybranych architektur wyjaśniono różnice pomiędzy nimi. Omówiono zagadnienia różnej świadomości dotyczące architektur wykazywanych przez informatyków pełniących różne role i o potencjalnych skutkach ignorowania architektury. Opisano różnice pomiędzy często mylonymi pojęciami takimi jak architektury, wzorce architektoniczne i projektowe. Na końcu zawarta jest konkluzja, że stare zagadnienie transformowania modeli architektury w działające oprogramowanie może się powieść teraz z wykorzystaniem modeli architektur i rozwiązań sztucznej inteligencji. Znaczenie modeli architektonicznych się umocni, gdyż będą one podstawą do generowania oprogramowania.

SŁOWA KLUCZOWE

Architektura, Rodzaje architektur IT, Kompetencje architektoniczne, Wzorce architektoniczne i projektowe.

Wprowadzenie

Architekturę znamy dosyć dobrze z projektów budowlanych. W przypadku budowy domku jednorodzinnego chcemy by dom był funkcjonalny, estetyczny, bezpieczny, łatwy w utrzymaniu. Jednak możemy narzekać, że architekt zapowiedział, że dom ma ograniczenia co do wysokości i nie może być budowany zbyt blisko granic działki, a instalacje i ich podłączenia do źródeł zewnętrznych nakładają szereg ograniczeń. Dodatkowo dom powinien spełniać wymogi energetyczne.

Zatem co to jest ogólnie architektura? Architektura to sztuka i nauka planowania, projektowania oraz organizowania przestrzeni, struktur lub systemów, tak aby spełniały określone wymagania funkcjonalne, estetyczne i techniczne.

Wyróżnia się następujące podstawowe cechy architektury [1]:

1. Struktura: Sposób, w jaki elementy systemu są rozmieszczone i jak są ze sobą powiązane.
2. Bloki konstrukcyjne (Building Blocks): W każdym systemie wyróżnia się jednostki pełniące określone funkcje przy budowie całego systemu.
3. Zasady projektowe: Zestaw zasad panujących nad procesem projektowania i rozwoju, monitorowaniem i utrzymaniem zbudowanego obiektu.
4. Cele: Architektura ma określone cele, którymi są: funkcjonalność, estetyka, skalowalność, wydajność, bezpieczeństwo, zgodność z wymaganiami użytkowników, itp.
5. Dla przykładu cechami architektury budowlanej będą:
6. Struktura: Układ przestrzenny, np. rozmieszczenie pomieszczeń w budynku; Konstrukcja nośna; Układ komunikacyjny, np. korytarze, schody, windy i wejścia; Strefy funkcjonalne.
7. Bloki konstrukcyjne: Fundamenty, ściany nośne, dach, stropy, okna i drzwi, instalacje techniczne, elewacja.
8. Zasady projektowe:

- Trwałość i bezpieczeństwo: Trwałość i odporność materiałów, stabilna konstrukcja nośna spełniająca normy, analiza inżynieryjna wytrzymałości.
- Funkcjonalność i dostępność: Układ przestrzenny, dostępność i ergonomia.
- Zrównoważony rozwój i efektywność energetyczna: Izolacja termiczna, odnawialne źródła energii, zarządzanie wodą, materiały przyjazne dla środowiska.
- Monitorowanie i konserwacja: Łatwy dostęp do elementów konstrukcyjnych, zintegrowane systemy monitorowania, planowanie konserwacji i remontów, materiały i technologie ułatwiające utrzymanie.
- Cele: Zapewnienie bezpieczeństwa, funkcjonalności, trwałości, estetyki, efektywności energetycznej, zrównoważonego rozwoju oraz łatwego utrzymania i monitorowania.

Właściwości architektury budowlanej możemy przenieść na grunt architektur IT. Podobnie jak w budownictwie, architektury IT mają na celu zapewnienie solidnych fundamentów, na których można zbudować funkcjonalne i skalowalne systemy. Autor pracy jako były programista uczestniczył w budowie wielu systemów informatycznych. Raz przy budowie sporego systemu informatycznego kierownictwo firmy wręcz nakazało tworzenie prowizorycznych rozwiązań by przyspieszyć rozwój i oddanie oprogramowania. Ten sposób działania odniósł sukces na krótką metę, gdyż system został wdrożony. Natomiast w dłuższej perspektywie musiano wprowadzać wiele kosztownych poprawek związanych z początkowymi prowizorycznymi działaniami, o których kierownictwo wolało nie pamiętać.

Typy architektury związanych z IT

Informatycy w zależności od stanowiska mówią bardzo często o różnych architekturach. W zależności od obszaru zastosowania lub użytych technologii możemy wyróżnić architektury:

1. Architektura Biznesowa (Business Architecture): Dotyczy struktury biznesowej organizacji, w tym procesów biznesowych, organizacji pracy, modeli biznesowych i produktów. Wyznacza cele biznesowe, motywację i strategię działania.

2. Architektura Informacyjna (Information Architecture): Koncentruje się na zarządzaniu i przepływie informacji w organizacji, obejmując modelowanie, organizację i cykl życia danych.

3. Architektura Aplikacji (Application Architecture): Określa sposób projektowania, integracji i zarządzania aplikacjami w organizacji, w tym podział na warstwy (prezentacja, logika biznesowa, dane).

4. Architektura Oprogramowania (Software Architecture): Skupia się na strukturze i organizacji kodu, obejmując podział na moduły, klasy, komponenty oraz wzorce projektowe (np. mikroserwisy, MVC).

5. Architektura Danych (Data Architecture): Obejmuje strukturę, zarządzanie i przechowywanie danych w organizacji, w tym modele danych i strategie przechowywania (np. Big Data).

6. Architektura Technologiczna (Technology Architecture): Definiuje infrastrukturę technologiczną, w tym sprzęt, oprogramowanie i sieci, potrzebne do realizacji celów organizacji.

7. Architektura Integracji (Integration Architecture): Dotyczy sposobów integracji systemów, aplikacji i danych w organizacji, obejmując różne narzędzia i metodyki (np. middleware, ESB).

8. Architektura Systemu (System Architecture): Obejmuje projektowanie i organizację całych systemów informatycznych, ich komponentów i sposobu integracji.

9. Architektura Bezpieczeństwa (Security Architecture): Skupia się na zabezpieczeniu informacji, systemów i procesów w organizacji poprzez użycie polityk bezpieczeństwa i mechanizmów ochronnych.

10. Architektura Sieciowa (Network Architecture): Zajmuje się projektowaniem i zarządzaniem infrastrukturą sieciową, umożliwiającą komunikację między systemami i urządzeniami (np. LAN, WAN).

11. Architektura Chmury (Cloud Architecture): Obejmuje projektowanie i organizację zasobów IT w chmurze, w tym zarządzanie usługami SaaS, PaaS i IaaS oraz integracją chmury z lokalną infrastrukturą.

Dodatkowo możemy wyróżnić pojemną pojęciowo architekturę korporacyjną (Enterprise Architecture (EA). Architektura ta [2] [3] bazuje na strukturalnym podejściu, które integruje w spójną całość różne aspekty działalności organizacji począwszy od strategii, biznesu, poprzez aplikacje, technologie, po dane. EA obejmuje architekturę biznesową, architekturę informacyjną, architekturę aplikacji i architekturę technologiczną. Dzięki EA [4] organizacja może nie tylko dostosować się do zmian w otoczeniu, ale także optymalizować zasoby ludzkie, finansowe i technologiczne, eliminując duplikacje, redukując koszty oraz zarządzając ryzykiem i zgodnością z regulacjami. EA wspiera również strategiczne zarządzanie i ciągłe doskonalenie, pomagając organizacji w realizacji długoterminowych celów.
Należy zaznaczyć, że za różne architektury odpowiadają odpowiedni specjaliści mający odpowiednie kompetencje. Współpraca tych fachowców pozwoli na tworzenie wysokiej jakości kompleksowej architektury IT.

Architektury o podobnych nazwach

Niekiedy wydaje się, że do tej samej architektury stosowane są podobne nazwy. Okazuje się jednak, że pomimo podobnych nazw chodzi o różne architektury. W tym rozdziale wybrano trzy przykłady.
Architektura informacji i architektura danych wydają się być podobne. Architektura informacji (IA) koncentruje się na organizacji, strukturze i przepływie informacji w organizacji, co jest kluczowe dla jej funkcjonowania na poziomie biznesowym. Architektura danych natomiast koncentruje się na technicznych aspektach zarządzania danymi, takich jak modelowanie danych, zarządzanie bazami danych, przepływy danych, ich przechowywanie i przetwarzanie. Jest to szczególnie ważne w projektach związanych z hurtowniami danych, gdzie struktura danych musi być zoptymalizowana pod kątem wydajności i analizy.

W świecie IT często mówi się o architekturze systemowej i architekturze rozwiązania (Solution Architecture). Architekturę rozwiązania można uznać za specyficzny przypadek architektury systemowej, który dotyczy konkretnego zastosowania lub rozwiązania technologicznego w określonym kontekście biznesowym. Architektura rozwiązania jest bardziej praktyczna i ukierunkowana na dostarczenie działającego rozwiązania, które spełnia określone wymagania. Przykładem może być zaprojektowanie architektury systemu e-commerce, który integruje platformę płatności, zarządzanie zapasami i system wysyłek. Architektura systemowa jest szerszym pojęciem, obejmującym projektowanie i organizację komponentów systemu informatycznego w sposób, który zapewnia jego skalowalność, wydajność i bezpieczeństwo. Jest to fundament, na którym opierają się poszczególne rozwiązania, takie jak wspomniane rozwiązania e-commerce.

Wreszcie czy architektura aplikacji to nie to samo co architektura oprogramowania? Architektura aplikacji skupia się na projektowaniu i organizacji konkretnej aplikacji oraz jej integracji z innymi systemami, tak aby spełniała określone wymagania biznesowe. Przykładem może być architektura aplikacji CRM, która musi integrować się z systemem ERP i aplikacją do obsługi klienta. Architektura oprogramowania jest bardziej ogólnym podejściem, obejmującym projektowanie całego systemu oprogramowania, w tym jego aspekty techniczne i strukturalne, niezależnie od liczby aplikacji wchodzących w skład systemu. Przedstawmy bardziej szczegółowo istotne własności architektury aplikacji i oprogramowania. Najpierw należy wyjaśnić pojęcia modułu i komponentu. Moduł jest jednostką organizacyjną kodu źródłowego o określonej funkcjonalności, a komponent jest jednostką wykonawczą ,która może być łatwo wymieniana i integrowana z innymi komponentami w systemie.

Jeśli chodzi o strukturę to architektura aplikacji będzie dotyczyć aspektów:
- Układ modułów: Każdy moduł ma określoną funkcjonalność, a komunikacja między nimi jest właściwie zdefiniowana.
- Interakcje i komunikacja: Moduły aplikacji komunikują się ze sobą, np. poprzez interfejsy API, usługi sieciowe lub komunikaty asynchroniczne.
- Skalowalność i elastyczność: Dodawanie nowych funkcji powinno być łatwe, a zmiana obciążenia nie powinna prowadzić do problemów związanych z wykonaniem oprogramowania.
- Użyteczność: Aplikacja powinna być łatwa w obsłudze, intuicyjna i spełniać oczekiwania użytkowników końcowych.

Natomiast struktura architektury oprogramowania będzie powiązana z zagadnieniami:
- Warstwy i komponenty: Definiowany jest podział systemu na warstwy oraz komponenty realizujące określone funkcje.
- Wzorce projektowe: Stosowane są konkretne wzorce projektowe, takie jak MVC (Model-View-Controller) czy mikroserwisy, które pomagają organizować i zarządzać kodem.
- Zarządzanie zależnościami: Określenie zależności pomiędzy komponentami systemu, co ułatwi zarządzanie zmianami i aktualizacjami.
- Spełnienie atrybutów jakości takich jak: dostępność, wydajność, bezpieczeństwo i inne.

W przypadku bloków konstrukcyjnych architektura aplikacji obejmuje konkretne funkcje biznesowe realizowane poprzez:
- Warstwę prezentacji (UI): Odpowiada m.in. za interakcje użytkownika z aplikacją, prezentowanie danych.
- Warstwę logiki biznesowej: Zawiera zasady i reguły, które rządzą przetwarzaniem danych i operacjami w aplikacji.
- Warstwę dostępu do danych: Zarządzanie operacjami na danych, w tym interakcjami z bazami danych, systemami plików i usługami zewnętrznymi.
- Usługi pomocnicze: Obejmuje funkcje takie jak: uwierzytelnianie, zarządzanie sesjami, monitorowanie, oraz integracje z zewnętrznymi systemami i API.

Natomiast bloki konstrukcyjne architektury oprogramowania będą opisane przez:
- Komponenty kodu: Każdy komponent oprogramowania pełni określoną funkcję i może być niezależnie rozwijany, testowany i wdrażany.
- Bazy danych: Komponenty zarządzające przechowywaniem i dostępem do danych kluczowe dla działania aplikacji.
- Interfejsy API: Zbiory funkcji, które umożliwiają komunikację między różnymi częściami oprogramowania oraz z zewnętrznymi systemami.
- Usługi infrastruktury: Obejmują funkcje takie jak: logowanie, autoryzacja, monitorowanie i inne.

Zasady projektowe dla architektury aplikacji dotyczą:
- Modularności: Aplikacja jest podzielona na moduły, które są niezależne i łatwe do zarządzania, co ułatwia rozwój, testowanie i wdrażanie nowych funkcji.

- Reużywalności: Komponenty aplikacji są projektowane w taki sposób, aby mogły być ponownie wykorzystywane.
- Bezpieczeństwa: Budowa mechanizmów ochrony danych i użytkowników, takie jak: szyfrowanie, autoryzacja i uwierzytelnianie.
- Wydajności: Aplikacja jest zaprojektowana z myślą o optymalnej wydajności, minimalizacji opóźnień i maksymalizacji efektywności zasobów.

Natomiast zasady projektowe architektury oprogramowania to:
- Podział odpowiedzialności (Separation of Concerns): Każdy komponent oprogramowania ma wyraźnie zdefiniowaną odpowiedzialność, co minimalizuje ryzyko błędów i ułatwia zarządzanie systemem.
- Reużywalność kodu: Kod jest pisany w sposób, który umożliwia jego ponowne użycie w różnych częściach systemu lub w innych projektach.
- Elastyczność: Architektura jest zaprojektowana tak, aby umożliwiać łatwe wprowadzanie zmian i dostosowywanie oprogramowania do nowych wymagań.
- Bezpieczeństwo: Oprogramowanie jest projektowane z myślą o bezpieczeństwie, z uwzględnieniem najlepszych praktyk w zakresie ochrony przed zagrożeniami.

Głównymi celami architektury aplikacji są:
- Zaspokojenie potrzeb użytkowników: Zapewnienie, że aplikacja jest funkcjonalna, intuicyjna i spełnia wymagania użytkowników.
- Łatwość utrzymania i rozwoju: Architektura aplikacji ułatwia wprowadzanie zmian, aktualizacje oraz rozwiązywanie problemów technicznych.
- Zabezpieczenie danych i procesów: Zapewnienie wysokiego poziomu bezpieczeństwa, chroniącego aplikację i jej użytkowników przed zagrożeniami.
- Efektywność operacyjna: Aplikacja jest zoptymalizowana pod kątem wydajności i efektywności, co umożliwia jej sprawne działanie przy minimalnym zużyciu zasobów.

Cele architektury oprogramowania obejmują:
- Stabilność i niezawodność: Zapewnienie, że oprogramowanie działa stabilnie i jest odporne na błędy oraz awarie.
- Skalowalność: Możliwość rozbudowy systemu w odpowiedzi na rosnące potrzeby użytkowników bez konieczności przeprojektowywania całej architektury.
- Łatwość utrzymania: Architektura ułatwia zarządzanie kodem, naprawę błędów, oraz implementację nowych funkcji.
- Efektywność: Zapewnienie, że system działa wydajnie, zużywa minimalne zasoby i spełnia wymagania użytkowników pod kątem czasu odpowiedzi i przetwarzania.

Należy podkreślić, że architektura aplikacji jest bardziej bezpośrednio powiązana z biznesowymi wymaganiami i doświadczeniami użytkowników, natomiast architektura oprogramowania zajmuje się bardziej technicznymi aspektami i długoterminową stabilnością oraz elastycznością systemu. Zwykle za architekturę aplikacji jest odpowiedzialny architekt rozwojowy (solution architekt), to za za architekturę oprogramowania będzie zwykle odpowiedzialny architekt systemowy lub korporacyjny.
Niektóre architektury wydają się mieć podobne nazwy, lecz dokładniejsza analiza pokazuje, że istotnie się różnią funkcją lub szczegółowością opisu rozwiązań.

Świadomość architektury IT wśród programistów

Wielu programistów po prostu koduje nie dbając lub nie będąc świadomymi architektury na której bazują. Powody tej sytuacji mogą być następujące:

1. Szybkie dostarczenie produktu (MVP w metodykach zwinnych): W trakcie rozwoju oprogramowania często kładzie się nacisk na szybkie iteracje i dostarczanie wartości, architektura może być postrzegana jako zbyt sztywna lub nawet jako przeszkoda. Sytuacja ta prowadzić do powstania długu technicznego, którym będzie trzeba zarządzać [5].
2. Brak wiedzy na temat znaczenia architektury: Brak edukacji i zrozumienia zasad architektonicznych jest kluczowym problemem. Programiści, którzy nie rozumieją długoterminowych korzyści wynikających z dobrej architektury, mogą ignorować jej znaczenie, koncentrując się na bezpośrednich zadaniach programistycznych. Dodatkowo szczególnie młodzi programiści mogą nie mieć pełnego obrazu całej architektury oprogramowania.
3. Dodatkowy wysiłek związany z przestrzeganiem zasad architektonicznych: Stosowanie zasad architektonicznych często wymaga dodatkowego kodowania, testowania i dokumentowania, co może być postrzegane jako obciążenie, szczególnie w sytuacjach, gdzie priorytetem jest szybkość dostarczania.
4. Brak wsparcia kierownictwa i architektów: Przy braku wsparcia ze strony kierownictwa i architektów, dbałość o architekturę może zostać zepchnięta na dalszy plan. Jeśli organizacja nie kładzie nacisku na architekturę, programiści mogą czuć, że nie warto poświęcać na nią dodatkowego czasu i energii.
5. Brak widocznych korzyści w krótkiej perspektywie: Korzyści z dobrej architektury są często długoterminowe, co może być trudne do zauważenia na wczesnych etapach projektu. Programiści, którzy nie widzą natychmiastowych efektów swojej pracy, mogą zniechęcić się do przestrzegania zasad architektonicznych.
6. Brak długoterminowego spojrzenia: W codziennej pracy programiści mogą być bardziej skupieni na krótkoterminowych celach, takich jak ukończenie konkretnego zadania, czy iteracji. Brak perspektywy długoterminowej może prowadzić do ignorowania znaczenia solidnej architektury, która jest kluczowa dla przyszłego utrzymania i rozwoju systemu.
7. Gotowe frameworki z wbudowanymi rozwiązaniami architektonicznymi: Korzystanie z frameworków może tworzyć fałszywe poczucie bezpieczeństwa, że wszystkie problemy architektoniczne są już rozwiązane. Jednakże frameworki mają swoje ograniczenia i niewłaściwe ich użycie może prowadzić do problemów, które byłyby łatwiejsze do uniknięcia przy bardziej przemyślanej architekturze. Programistów często cechuje brak spojrzenia na długoterminowe utrzymanie i rozwój oprogramowania wymagających dobrych rozwiązań architektonicznych.

Aby zwiększyć świadomość architektury wśród programistów, kluczowe jest zainwestowanie w edukację, transparentność procesów projektowania architektury, oraz stworzenie kultury, w której architektura jest uznawana za istotny element rozwoju oprogramowania. Odpowiedzialność za naprawienie sytuacji dotyczy wielu ról, a w szczególności: architektów IT za definiowanie i komunikowanie architektury, liderów i kierownictwo techniczne oraz samych programistów, wspieranie rozwiązań architektonicznych. Kluczowe jest, aby wszyscy ci

uczestnicy współpracowali, promując kulturę techniczną, w której architektura jest integralną częścią procesu tworzenia oprogramowania. W wielu pozycjach literaturowych proponowane są rozwiązania mające poprawić efektywność rozwoju architektury [6], [7].
Różnice pomiędzy architekturami, wzorcami architektonicznymi i projektowymi

Przy rozwoju dobrze zbudowanej architektury należy pamiętać o wyznaczaniu atrybutów jakości oprogramowania oraz o wzorcach architektonicznych i projektowych.

Architektury dotyczą konkretnego obszaru działania i technicznych rozwiązań stosowanych w systemach IT. Definiują strukturę i organizację poszczególnych elementów systemu w określonym kontekście technologicznym lub funkcjonalnym. Przykłady tychże architektur podano wcześniej od architektury biznesowej do chmurowej.
Wzorce architektoniczne dotyczą sprawdzonych metod i podejść do rozwiązywania określonych problemów technicznych na poziomie architektury systemu. Przykładami takich wzorców architektonicznych są: Client-Server (Klient-Serwer), Warstwowa (Layered), Zdarzeniowa (Event-Driven), Peer-to-Peer (P2P) i inne.

Wzorce projektowe (design patterns) to sprawdzone rozwiązania dla typowych problemów projektowych, które mogą pojawić się na poziomie implementacji kodu. Przykładami wzorców mogą być: Singleton zapewniający, że dana klasa ma tylko jedną instancję i udostępnia ją globalnie; Observer, w którym obiekt (obserwowany) informuje zarejestrowane obiekty (obserwatorów) o zmianach stanu.
Ważne jest by rozwijając oprogramowanie posiadać wiedzę z zakresu oceny jakości działania jak i posiadać dobre rozeznanie z wzorców architektonicznych i projektowych.

Podsumowanie

Transformowanie modeli architektury w działające oprogramowanie to koncepcja, która ma już długą historię, jak choćby przykład języka UML, który od lat służy do opisu funkcjonowania oprogramowania. Mimo to, automatyczne przekształcanie modeli EA w działające systemy wciąż jest dynamicznie rozwijającą się dziedziną. Współczesne platformy Low-Code/No-Code zaczynają wykorzystywać AI do transformacji modeli architektonicznych i biznesowych (w tym BPMN) w działające aplikacje, co otwiera nowe możliwości. W miarę rozwoju sztucznej inteligencji i zaawansowania tych narzędzi, możemy spodziewać się powstania jeszcze bardziej efektywnych platform, które pozwolą na bezpośrednie przekształcanie modeli w gotowe systemy informatyczne. To oznacza, że znaczenie architektury oprogramowania będzie rosło, gdyż stanie się ona kluczowym narzędziem umożliwiającym szybkie i skuteczne tworzenie kompleksowych systemów.

Bibliografia

[1] Bass L, Clements P, Kazman R. (2012) Software Architecture in Practice, 3 edition. Addison-Wesley Professional, Upper Saddle River, NJ
[2] Lankhorst M. (2017) Enterprise Architecture at Work. Springer Berlin Heidelberg, Berlin, Heidelberg
[3] Werewka J, Jamróz K, Pitulej D. (2014) Developing Lean Architecture Governance at a Software Developing Company Applying ArchiMate Motivation and Business Layers. In: Kozielski S, Mrozek D, Kasprowski P, et al (eds) Beyond Databases, Architectures, and Structures. Springer International Publishing, Cham, pp 492–503
[4] The Open Group (2018) The TOGAF® Standard, Version 9.2. http://pubs.opengroup.org/architecture/togaf9-doc/arch/. Accessed 18 Aug 2018
[5] Li Z, Liang P, Avgeriou P (2015) Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios. In: 2015 12th Working IEEE/IFIP Conference on Software Architecture (WICSA). pp 65–74
[6] Martin RC (2017) Clean Architecture: A Craftsman's Guide to Software Structure and Design, 1st ed. Prentice Hall Press, USA
[7] Building Evolutionary Architectures[Book]. https://www.oreilly.com/library/view/building-evolutionary-architectures/9781491986356/. Accessed 1 Sep 2024.