**1. Bevezetés**

**1.1 Háttér**

A szolgáltatóipar digitalizációja az elmúlt évtizedben drámai mértékben felgyorsult, és mára egyre több vállalkozás ismeri fel az online eszközök nyújtotta előnyöket. Az autószervizek esetében a hagyományos telefonos vagy papíralapú időpontfoglalás már nem képes lépést tartani a vevők egyre magasabb elvárásaival, akik gyors, kényelmes és 0–24 elérhető megoldásokat kívánnak. Ezzel párhuzamosan a műhelyekben dolgozó szakemberek is egyre nagyobb terhelést tapasztalnak az manuálisan kezelt foglalások, módosítások és visszaigazolások miatt, ami a működés hatékonyságát jelentősen csökkenti.

A modern webes és mobil technológiák az elosztott rendszerek, a RESTful architektúra és a felhőalapú szolgáltatások integrálásával lehetőséget teremtenek arra, hogy a foglalási folyamat mind ügyfél-, mind üzemeltetői oldalon magabiztosan, automatizáltan és hibatűrően menjen végbe. A jelen projektben alkalmazott Laravel backend és React frontend ökoszisztéma ötvözi a gyors fejlesztési tempót, a könnyen karbantartható kódot és a skálázható adatbázis-kezelést, így ideálisan szolgálja az autószerviz speciális igényeit.

**1.2 Problémafelvetés**

A hagyományos időpontfoglalási módszerek számos nehézséget hordoznak magukban: az ütköző foglalások, a visszaigazolások kézi kezelése és az ügyfél-adatok biztonságos tárolása mind-mind olyan pontok, ahol emberi mulasztás vagy kommunikációs késlekedés akadályozhatja a zavartalan munkavégzést. A szervizekben sokszor hiányzik a valós idejű áttekintés, így a kapacitástervezés, a munkatársak beosztása és az egyes szolgáltatások elvégzése közötti koordináció pontatlan és kevéssé átlátható. Ez mind az ügyfélélményt, mind a belső erőforrás-gazdálkodást negatívan befolyásolja.

**1.3 Megoldási koncepció**

Az Autószerviz Időpontfoglaló Rendszer online, webes és mobilbarát felülettel szolgál az ügyfelek számára, lehetővé téve számukra, hogy bármikor, bárhonnan kiválaszthassák a kívánt szolgáltatást és időpontot. A REST API alapú Laravel backend biztosítja az adattároláshoz és -kezeléshez szükséges biztonságot, tranzakciókezelést és jogosultságkezelést, míg a React frontend egy dinamikus, reszponzív felhasználói élményt nyújt asztali és mobil környezetben egyaránt. A rendszer integrált SMS- és e-mail-értesítések küldésével automatikusan visszaigazolja és emlékezteti az ügyfeleket, míg a szerelők és recepciósok számára naptárszinkronizációs modul teszi áttekinthetővé a napi munkabeosztást.

**1.4 Projektcélok és hatókör**

A projekt elsődleges célja egy olyan teljes funkcionalitású szoftvercsomag elkészítése, amely az autószerviz napi működésének minden lényeges elemét lefedi. Ezen belül különösen fontos, hogy a fejlesztés során betartsuk a tiszta kód elveit, a dokumentáció pedig lehetővé tegye a későbbi bővítést és karbantartást. A hatókörbe tartozik a felhasználói regisztráció, járműnyilvántartás, szolgáltatáskatalógus-kezelés, időpontfoglalás, visszaigazolás, naptárszinkronizáció és a GDPR-kompatibilis adattárolás megvalósítása.

**1.5 Dokumentum felépítése**

Ez a dokumentum következetesen vezeti végig az olvasót a projekt minden aspektusán: először részletesen bemutatja a funkcionális és nem funkcionális követelményeket, majd pedig a rendszerterv részleteit, az architektúrák és adatmodellek megrajzolását. A fejlesztési ütemterv szakaszban pontos idő- és erőforrás-becsléseket találunk, ezt követi a moduláris megvalósítás ismertetése backend és frontend oldalon. A tesztelési fejezet dokumentálja a különböző tesztelési szinteket és az elért eredményeket, majd a telepítés és üzemeltetés leírása zárja a fő tartalmakat. A mellékletek között szerepel a forráskód elérhetősége, az adatbázis exportja, valamint a telepítőcsomagok és a felhasználói kézikönyvek.

## 2. Projektáttekintés

### 2.1 Projektcélok

A projekt elsődleges célja egy olyan online időpontfoglaló rendszer létrehozása, amely teljes mértékben digitalizálja az autószerviz foglalási folyamatait, megszünteti a telefonos és papíralapú adminisztrációs terheket, valamint 0–24 elérhetőséget biztosít az ügyfeleknek. A rendszernek lehetővé kell tennie a szolgáltatások (olajcsere, fékjavítás, műszaki vizsga stb.) egyszerű és átlátható kiválasztását, a járműadatok (rendszám, típus) rögzítését, és azonnali visszaigazolást SMS vagy e-mail formájában.

Ezen túlmenően a projekt arra törekszik, hogy a szerviz dolgozóinak valós idejű naptárszinkronizációval és terheléskiegyenlítéssel segítse a munkabeosztás optimalizálását, minimalizálja az ütköző foglalások számát, és ezáltal növelje a műhely hatékonyságát. A fejlesztés során kiemelt szerepet kap a tiszta kód irányelveinek betartása, a moduláris felépítés, valamint a könnyen karbantartható és bővíthető architektúra kialakítása.

### 2.2 Háttér és motiváció

Az autószervizek többsége még ma is hagyományos, telefonos vagy papíralapú időpontfoglalási módszert használ, ami jelentős adminisztratív többletmunkát és kommunikációs késedelmeket eredményez. A modern fogyasztók viszont egyre inkább elvárják az azonnali, felhasználóbarát digitális szolgáltatásokat, különösen mobil eszközökről, ahol a gyors hozzáférés alapkövetelmény.

A COVID-19 világjárvány idején felértékelődött az érintésmentes, online megoldások szerepe, ugyanakkor számos kis- és középvállalkozás még nem tette meg a szükséges technológiai ugrást. A digitális rendszerek bevezetése nemcsak a versenyképességet növeli, hanem csökkenti a hibalehetőségeket, optimalizálja az erőforrások kihasználtságát, és hosszú távon mérhető költségmegtakarítást is eredményez.

### 2.3 Érintett szereplők

Elsődleges szereplők a végfelhasználó ügyfelek, akik egyszerű, gyors és mobilbarát felületet igényelnek, valamint a szervizben dolgozó szerelők és recepciósok, akik áttekinthető naptárnézetre és rugalmas beosztási lehetőségekre támaszkodnak a mindennapi munkavégzés során. A projektért felelős fejlesztőcsapat tagjai közé tartoznak a backend- és frontend-fejlesztők, az adatbázis-szakértők, valamint a tesztelők, akik együttesen biztosítják a rendszer megbízhatóságát és skálázhatóságát.

Másodlagos szereplők az SMS- és e-mail-értesítéseket kezelő szolgáltatók, a naptárszinkronizációs API-k, valamint a GDPR-megfelelésért és adatbiztonságért felelős jogi tanácsadók. A jövőbeli karbantartást és bővítést végző csapatok szintén kulcsszerepet játszanak abban, hogy a rendszer hosszú távon is fenntartható és rugalmas maradjon.

## 3. Funkcionális követelmények

### 3.1 Online időpontfoglalás

Az online időpontfoglaló felület célja, hogy a felhasználók bármikor, regisztráció után vagy vendégként pillanatok alatt válasszanak szabad időpontot. A foglalási űrlap real-time lekérdezi az adatbázist a már lefoglalt és foglalható időpontokról, majd egy dinamikusan frissülő naptárnézetben mutatja meg az ügyfél számára elérhető intervallumokat. A rendszer a foglalás véglegesítése előtt ellenőrzi az időpont érvényességét és párhuzamos tranzakciókezelést alkalmaz, hogy elkerülje az ütközéseket, majd sikeres lefoglalás esetén azonnal rögzíti az adatot.

A foglalási folyamat során az ügyfél kiválasztja a kívánt dátumot és időintervallumot, majd megadja kapcsolattartási adatait. Hibás vagy hiányos adatok esetén a felület valós idejű visszajelzést nyújt, és pontosan jelzi, mely mezőket szükséges javítani. A rendszer skálázhatósága lehetővé teszi, hogy egyszerre több száz párhuzamos foglalási kérést kezeljen, minimalizálva a válaszidőt és biztosítva az 0–24 órás rendelkezésre állást.

### 3.2 Szolgáltatások kiválasztása

A szolgáltatáskatalógus a felhasználó számára részletes leírással, becsült időtartammal és árral jelenik meg. Amikor az ügyfél időpontot foglal, többféle szolgáltatás (például olajcsere, fékjavítás, műszaki vizsga) közül választ, és akár egyszerre több tételt is összevonhat egyetlen látogatásra. A frontenden a szolgáltatások checkbox-okkal vagy jelölőlistával kezelhetők, a backend pedig relációs táblában tárolja a szolgáltatásneveket, leírásokat, idő- és díjértékeket, valamint az egyes szolgáltatások egymásra épülhetőségét.

A rendszer figyelembe veszi, hogy egyes szolgáltatások speciális munkaidőt vagy szakember-hozzárendelést igényelnek, és ennek megfelelően zárja ki a foglalható időablakokat. Ha a kiválasztott szolgáltatások összesített időtartama meghalad egy szabad intervallumot, a felület alternatív időpontokat kínál fel, ezzel is segítve a felhasználót a gyors döntésben.

### 3.3 Járműadatok kezelése

A felhasználók fiókjukban önállóan vehetnek fel új járműveket, illetve szerkeszthetik a korábban rögzítetteket. Egy jármű esetén kötelező mezők a rendszám, a gyártó és modell, a gyártási év, valamint opcionálisan a jármű alvázszáma. A rendszer formális ellenőrzéseket végez a rendszám formátumára és a gyártási év ésszerűségére, hogy a rossz adatbevitel elkerülhető legyen.

Minden foglaláshoz egy vagy több járműhöz tartozó adatok kapcsolódnak, így a szerviz szakemberei könnyen áttekinthetik az előzményeket, szervizelőzményeket és a garanciális státuszt. A backend relációs adatmodellben kapcsolja össze az ügyfél- és járműadatokat, támogatva a több gépjármű regisztrációját egy felhasználói fiók alatt.

### 3.4 Értesítések (SMS/e-mail)

A rendszer automatikusan generál és küld értesítéseket a foglalás állapotáról: visszaigazoláskor és az időpont előtt meghatározott idővel (például 24 illetve 2 órával korábban) emlékeztető SMS-t és e-mailt továbbít. A küldési folyamat külső szolgáltató API-in keresztül valósul meg, ahol hibás küldés esetén többszöri újrapróbálkozás és végső soron fallback e-mail alkalmazása biztosítja, hogy az ügyfél mindenképpen tájékozódjon.

A sablonrendszer lehetővé teszi a személyre szabott üzenetek kialakítását, tartalmazza az ügyfél nevét, a szolgáltatás típusát, az időpont részleteit és egy visszaigazoló linket. A backend naplózza a sikeres és sikertelen küldéseket, ezekből automatikusan generált riportok futnak éjszaka a szervizvezetők számára, hogy a kommunikációs problémákat időben orvosolhassák.

### 3.5 Naptárszinkronizálás

A szerviz személyzete számára egy központi naptárnézet áll rendelkezésre, amely valós időben tükrözi az ügyféloldalon történt foglalásokat és módosításokat. A rendszer ICS-formátumban exportálható naptárfeedet nyújt, valamint kétirányú szinkronizációt támogat Google Calendar vagy Exchange szerverekkel. Így a recepciósok és szerelők saját eszközeiken is frissített beosztást láthatnak anélkül, hogy manuálisan exportálnának vagy importálnának.

A naptárszolgáltató API-k hívásai aszinkron módon, eseményvezérelt üzenetküldéssel frissítik a külső naptárakat minden új foglalás, törlés vagy időpontmódosítás után. A backendben ütemezett feladatok ellenőrzik a szinkronizációs státuszt, és problémák esetén értesítik a rendszergazdát.

## 4. Nem funkcionális követelmények

A nem funkcionális követelmények határozzák meg a rendszer általános minőségi, viselkedési és üzemeltetési elvárásait. Ezek a kritériumok kiegészítik a funkcionális követelményekben leírtokat, és biztosítják, hogy az Autószerviz Időpontfoglaló Rendszer ne csak működjön, hanem gyors, megbízható, könnyen kezelhető és hosszú távon karbantartható legyen. A következőkben részletesen kifejtjük a legfontosabb nem funkcionális elvárásokat, azok mérőszámait, elfogadási kritériumait és megvalósítási stratégiáit.

### 4.1 Biztonság és adatvédelem (GDPR)

A rendszer teljes kommunikációja TLS 1.2 vagy magasabb verziójú HTTPS csatornán zajlik, hogy az ügyfél- és járműadatok titkosítva legyenek az átvitel során. A backendben minden érzékeny mező (jelszó, személyes adatok) AES-256 szabvánnyal titkosítva tárolódik, és csak a szerver oldali komponensek férhetnek hozzá a kulcsokhoz.

A felhasználói hitelesítést JWT tokenekkel valósítjuk meg, melyek rövid élettartamúak (30 perc), de automatikus frissítéssel támogatottak. Minden API-hívás a megfelelő jogosultsági szint ellenőrzésével (role-based access control) történik, így a recepciós, szerelő és adminisztrátor funkciók szigorúan elkülönülnek.

A GDPR-megfelelés érdekében a rendszer adatminimalizálást alkalmaz: csak azokat a mezőket kérjük be és tároljuk, amelyek a szolgáltatás nyújtásához feltétlenül szükségesek. A felhasználók bármikor kérhetik személyes adataik törlését vagy exportját, és a rendszer naplózza az adatkezelési eseményeket, hogy az auditálás minden lépése visszakövethető legyen.

### 4.2 Teljesítmény és válaszidő

Az API-válaszidők tekintetében a cél, hogy az egyszerű lekérdezések (pl. szabad időpontok lekérése) válaszideje ne haladja meg a 150 ms-ot, míg a komplexebb műveletek (foglalás, módosítás) esetén 300 ms alatti időt tűzünk ki. Frontenden a teljes oldalbetöltés első meaningful paint idejét 2 másodperc alatt tartjuk, mobilon pedig 3 másodperc alatt.

A backend szolgáltatások optimalizálása érdekében ORM-kihívásoknál előtöltést (eager loading) alkalmazunk, és kritikus lekérdezésekhez egyedi SQL-t vagy cache-elést (Redis) használunk. A frontend React-komponensek lazy loadinggal és kódszeleteléssel (code splitting) csökkentik a kezdeti betöltött csomagméretet.

A teljesítmény folyamatos monitorozását automatikus load tesztek támogatják: havonta futtatott JMeter- és k6-szkriptek ellenőrzik a legmagasabb forgalmi csúcsokat, és a CI/CD pipeline részeként érvényesítik, hogy a frissítések ne rontsák a válaszidőket.

### 4.3 Skálázhatóság és terheléskezelés

A szolgáltatások konténerezése Dockerben történik, és Kubernetes klaszterben futnak, így vízszintes skálázással könnyedén reagálunk a forgalomnövekedésre. A LoadBalancer automatikusan új példányokat indít, ha a CPU- vagy memóriahasználat 70 %-ot meghaladja.

Az adatbázis-kezelés shardinggal és read-replica architektúrával támogatja a nagy olvasási terhelést, míg az írási műveleteket master node kezeli. A replika késleltetését folyamatosan figyeli a rendszer, és kritikus szituációban a lekérdezéseket inkább a master node-ra irányítja.

Ez a felépítés lehetővé teszi, hogy akár több száz egyidejű foglalási és lekérdezési kérést is megbízhatóan kezeljünk anélkül, hogy a felhasználói élmény romlana.

### 4.4 Rendelkezésre állás és megbízhatóság

Az SLA 99,9 %-os éves rendelkezésre állást ír elő, ami legfeljebb 8,76 óra éves leállást enged meg. A rendszer magában foglal több availability zone-ban futó példányokat, így az egyes zónák kiesése sem okoz teljes kiesést.

Napi, automatizált biztonsági mentés készül mind a relációs adatbázisról, mind a konfigurációs fájlokról, és 24 órás RTO (recovery time objective) mellett képesek vagyunk visszaállítani a szolgáltatást. A deploy folyamat blue–green vagy canary verziózás mellett zajlik, minimalizálva a frissítések alatti kockázatot.

### 4.5 Használhatóság és hozzáférhetőség

A felhasználói felület mobil-first szemléletben készült, és megfelel a WCAG 2.1 AA szintű akadálymentességi követelményeknek: minden funkció elérhető billentyűzetről, képernyőolvasóval is használható, és megfelelő kontrasztú színeket alkalmazunk.

A tervezési irányelvek szerint a webes admin felület logikus menü- és űrlapszerkezetet követ, minimális kattintással vezetve végig az adminisztrátori feladatokon. A felhasználói visszacsatolások gyűjtése folyamatosan zajlik, és negyedévente UX-audit keretében értékeljük az ergonómiát, majd szükség szerint finomhangoljuk a komponenseket.

### 4.6 Karbantarthatóság és minőségi kód

A kód alapjául a SOLID elvek szolgálnak, modulonkénti felelősségi körökkel, egységtesztekkel és kódreview-kötelzettséggel. A SonarQube statikus kódelemzése biztosítja, hogy a frissítések ne rontsák a kód minőségét, és minden pull request automatikusan futtat lint-, formázási és biztonsági ellenőrzéseket.

A CI/CD pipeline-ban minden sikeres build deploy–teszt környezetbe kerül, ahol automatikus integrációs tesztek és end-to-end forgatókönyvek futnak. Csak ezek teljesülése után kerülhet sor a staging, majd éles környezetbe való átvezetésre, ezzel minimalizálva a hibák élő alkalmazásba kerülésének esélyét.

### 4.7 Kompatibilitás és hordozhatóság

A frontend a legelterjedtebb böngészőkön (Chrome, Firefox, Edge, Safari) garantáltan konzisztens élményt nyújt, és méretre optimalizált CSS–JS csomagokat építünk minden platformra. Mobilon iOS és Android eszközökön a webes kliens natív jellegű, reszponzív viselkedést mutat.

A backend platformfüggetlen Docker-konténerekben fut, így akár AWS, Azure vagy on-premise környezetben is telepíthető minimális konfigurációs módosítással. Az API-versioning segítségével a régi kliens-verziók továbbra is működőképesek maradnak új kiadás után is.

### 4.8 Naplózás, monitoring és auditálás

A szolgáltatások strukturált JSON formátumú naplózást alkalmaznak, melyeket központi log-gyűjtő (ELK stack) fogad és indexel. A Prometheus és Grafana eszközök folyamatosan gyűjtik a metrikákat (erőforrás-használat, válaszidők, hibaarány), és előre definiált küszöbök átlépésekor alert-eket küldenek a fejlesztőknek és üzemeltetőknek.

Az auditlog modul minden felhasználói műveletet időbélyeggel és szerepkörrel rögzít, így visszakövethető, ki, mikor és milyen változtatást hajtott végre. A logok legalább hat hónapig megőrződnek, ami megfelel a belső és jogi előírásoknak, és támogatja a biztonsági vizsgálatokat.

## 5. Rendszerterv és architektúra

### 5.1 Összetevők áttekintése

A rendszer három fő rétegből épül fel: a Laravel alapú REST API backendre, a React alapú frontend kliensre és a MySQL adatbázisra. A backend szolgáltatja az üzleti logikát, kezeli az autentikációt, az adatmanipulációkat és az értesítési mechanizmust. A frontend dinamikus, reszponzív felhasználói felületet biztosít, amely asztali és mobil böngészőkön egyaránt egységes élményt nyújt. A MySQL adatbázis tárolja az összes entitást – ügyfelek, járművek, szolgáltatások, foglalások, emlékeztetők és naptáresemények – normalizált struktúrában.

Minden komponens Docker-konténerekben fut, így a fejlesztés és az üzemeltetés közötti eltérés minimalizálható. A konténerek Kubernetes klaszterben helyezkednek el, automatikus skálázással és terheléselosztással, ami biztosítja a 0–24 elérhetőséget és a magas rendelkezésre állást. A CI/CD pipeline GitHub Actions segítségével kezeli az építést, tesztelést és kihelyezést, így az új verziók megbízhatóan kerülnek először teszt-, majd éles környezetbe.

### 5.2 Adatbázis-modelldiagram

Az adatmodell központi eleme az ügyfél (users), akihez több jármű (vehicles) kapcsolódik, és minden foglalás (bookings) egy adott járműhöz valamint egy vagy több szolgáltatáshoz (services) kötődik. A kapcsolatok 1:N és N:M típusúak, melyeket relációs táblák és idegen kulcsok valósítanak meg. Az alerts tábla tárolja az SMS–e-mail értesítések ütemezését és státuszát, míg a calendar\_events az ICS-exportolt naptáreseményeket kezeli.

A diagram harmadik normálforma szerint készült, hogy elkerüljük a redundáns adatokat és biztosítsuk az adatintegritást. Minden táblához primér kulcs tartozik, és az idegen kulcsokra indexek kerülnek a lekérdezési teljesítmény optimalizálása érdekében. A bookings táblában összetett kulcs biztosítja, hogy egy jármű ne legyen egyszerre több helyen foglalva, és a tranzakciókezelés ACID-előírásokat követ.

### 5.3 REST API specifikáció

Az API verziózása v1 útvonalprefixel történik, és minden kérést HTTPS protokollon keresztül autentikált JWT tokennel kell ellátni. A legfontosabb végpontok a következők:

* POST /api/v1/auth/login – belépés, token generálás
* GET /api/v1/services – szolgáltatáskatalógus lekérése
* POST /api/v1/bookings – új foglalás létrehozása
* GET /api/v1/bookings/{id} – foglalás részleteinek lekérdezése
* PUT /api/v1/bookings/{id} – foglalás módosítása
* DELETE /api/v1/bookings/{id} – foglalás törlése

A kérések és válaszok JSON-formátumúak, egységesen tartalmazzák a státuszmezőt és hibaeseteknél az ISO-szerinti hibakódot. A paginált listavégpontoknál limit–offset sémát alkalmazunk, és a rate limiting alapértelmezett korlátja 100 kérés/perc/felhasználó. Minden sikeres mutációs végpont 201-es státuszkóddal válaszol, és visszaadja az érintett erőforrás aktuális állapotát.

### 5.4 Frontend architektúra

A React alkalmazás moduláris, komponens-alapú felépítésű, ahol minden funkciós egység saját mappát kap: components, pages, hooks, services és styles. A globális állapotkezelést React Context és egyedi hookok biztosítják, míg komplex esetekben Redux-szcenáriók is alkalmazhatók. A routing React Router Dom segítségével valósul meg, ügyelve a védett (auth guard) és nyilvános útvonalak elkülönítésére.

A stíluskezelés CSS-modules vagy styled-components alapokon nyugszik, hogy elkerüljük az osztálynév-ütközéseket és támogassuk a témaváltást. A buildet Webpack végzi, amely kódszeletelést (code splitting) és lazy loadot alkalmaz a kezdeti betöltés felgyorsításához. A mobil-first megközelítésnek köszönhetően minden oldal rugalmasan alkalmazkodik a képernyőmérethez, a breakpointokat testreszabottan definiálva az admin és ügyfél nézetekre.

A frontend tesztelése Jest–React Testing Library tandemmel zajlik, ahol unit és integrációs tesztek mérik a felület stabilitását. A Cypress használatos end-to-end forgatókönyvekhez, amely során az éles szolgáltatással való együttműködést vizsgáljuk a foglalási, módosítási és értesítési folyamatok lefedettségével.

## 6. Fejlesztési ütemterv és erőforrások

A projekt fejlesztését egy hat sprintből álló, összesen 12 hetes Scrum-alapú módszertannal tervezzük, amely biztosítja a folyamatos visszajelzést, a rugalmasságot és a határidők betartását. Minden sprint két hétig tart, és a Sprint Planning, napi stand-upok, Sprint Review és Sprint Retrospective események keretében kerül lebonyolításra. A Sprint Backlog felépítése során a prioritás az üzleti érték maximalizálása, a kockázatok csökkentése és a korai működő képes verziók bemutatása.

Minden sprint kezdetén a Product Owner bemutatja a következő iteráció célját, és a csapat együttesen becsüli a feladatokat story pointokban. A Definition of Done dokumentumban rögzítettük, hogy egy feladat csak akkor zárható, ha a kód le van tesztelve, a dokumentáció frissítve, és a QA visszajelzései alapján az esetleges hibákat orvosoltuk. A folyamatszabályozás része továbbá a kódreview, a SonarQube-ellenőrzés és a CI/CD pipeline automatikus deploy-ja tesztkörnyezetbe minden merge után.

### 6.1 Módszertan és folyamatszabályozás

A fejlesztés Scrum keretrendszerben zajlik, amelynek sarokkövei:

1. Sprint Planning Minden sprint első munkanapján kétórás tervező meeting, ahol a csapat kiválasztja a következő két hét során megvalósítandó backlog elemeket, és felosztja azokat feladatokra.
2. Napi stand-upok Minden reggeli, legfeljebb 15 perces online értekezleten a csapattagok megosztják, min dolgoztak, mi az akadály, és mit terveznek ma befejezni.
3. Sprint Review A sprint utolsó napján bemutatjuk az elkészült funkciókat a megrendelőnek és az érintetteknek, begyűjtjük a visszajelzéseket, és frissítjük a Product Backlogot.
4. Sprint Retrospective A Review után egyórás megbeszélés a csapat számára, ahol áttekintjük, mi ment jól, mi ment kevésbé jól, és definiáljuk a következő sprintben alkalmazandó folyamatjavításokat.

A kódminőséget SonarQube ellenőrzések, lintelés és peer review biztosítja, míg a tesztelést automatizált unit, integrációs és end-to-end tesztek támogatják. A CI/CD folyamat GitHub Actions segítségével építi, teszteli és deploy-olja a kódot, minimalizálva az emberi beavatkozásból eredő hibákat.

### 6.2 Milestone-ok és időbecslések

A projekt 12 hetes ütemtervében a következő fő mérföldköveket (milestone) különítettük el, minden sprint végén működő demonstrációval:

1. Sprint 0 (2 hét) – Környezet előkészítés és architektúra-tervezés
   * Docker/Kubernetes klaszter felállítása: 3 nap
   * Projekt repository és CI/CD pipeline alapok: 2 nap
   * Adatbázis-modelldiagram és API specifikáció véglegesítése: 5 nap
   * Dokumentáció sablonok és kódminőségi szabályok definiálása: 2 nap
2. Sprint 1 (2 hét) – Felhasználói kezelés és autentikáció
   * JWT-alapú belépés és regisztráció backend implementálása: 4 nap
   * Frontend login/logout, regisztrációs formák: 3 nap
   * Unit tesztek írása és CI pipeline tesztelés: 3 nap
   * Review, bugfix és dokumentáció frissítés: 2 nap
3. Sprint 2 (2 hét) – Szolgáltatáskatalógus és járműadatok
   * services és vehicles CRUD API-k és adatbázis-táblák: 5 nap
   * Frontend szolgáltatás- és járműkezelő modulok: 4 nap
   * Integrációs tesztek és UI validációk: 3 nap
4. Sprint 3 (2 hét) – Foglalási modul és időpontkezelés
   * bookings API többszálú ütközéskezeléssel: 5 nap
   * Dinamikus naptárnézet és foglaló űrlap React-ben: 5 nap
   * Teljesítményoptimalizálás (cache, indexek): 2 nap
5. Sprint 4 (2 hét) – Értesítések és naptárszinkronizáció
   * SMS/e-mail küldő szolgáltatás integráció, retry mechanizmus: 4 nap
   * ICS-export és Google Calendar API szinkronizáció: 4 nap
   * E2E tesztek írása és validálás: 4 nap
6. Sprint 5 (2 hét) – Tesztelés, stabilizáció és dokumentáció
   * Teljes end-to-end tesztcsomag Cypress-szel: 4 nap
   * Load tesztek futtatása és eredmények értékelése: 3 nap
   * Felhasználói kézikönyv és fejlesztői dokumentáció összeállítása: 5 nap

Minden sprint végén demonstrációt tartunk, ahol a Product Owner visszaigazolja a teljesített feladatokat, és jóváhagyja a továbblépést. A mérföldkövek után legalább egy napot hagyunk az esetleges sprintvégi csúszások rendezésére.

### 6.3 Csapatfeladatok és kompetenciák

A fejlesztői csapat tagjai a következő szerepköröket és felelősségi köröket töltik be:

* Backend fejlesztő (1–2 fő) Felelős a Laravel REST API-k, az adatbázis-sémák és a szerveroldali logika kialakításáért. Erőforrás: napi 6–8 fejlesztői óra.
* Frontend fejlesztő (1–2 fő) A React alapú felület megalkotása, a reszponzív dizájn és a felhasználói interakciók implementálása tartozik hozzá. Erőforrás: napi 6–8 fejlesztői óra.
* DevOps mérnök (0,5–1 fő) Konténerizálás, CI/CD pipeline, Kubernetes klaszter karbantartás és deployment stratégiák kialakítása. Erőforrás: heti 10–15 óra.
* QA mérnök (0,5–1 fő) Tesztelési stratégiák, automatizált és manuális tesztek tervezése, végrehajtása, valamint hibajegyek kezelése. Erőforrás: sprintenként 15–20 tesztóra.
* Scrum Master / Projektmenedzser (1 fő) A folyamatok betartása, a sprintceremóniák vezetése, akadályok eltávolítása, erőforrás-allokáció. Erőforrás: heti 8–10 óra.

A csapat kompetenciái között szerepel a PHP és JavaScript modern keretrendszereinek mély ismerete, REST API tervezés, React Hookok és state management, Docker–Kubernetes üzemeltetés, valamint a Tesztautomatizálás eszközeinek (Jest, Cypress, JMeter) gyakorlatban szerzett tapasztalata. A projekt során elvárt a folyamatos tudásmegosztás, párprogramozás és peer code review, hogy minden csapattag ismerje a rendszer egészét, és könnyen át tudja venni mások feladatait.

## 7. Részletes megvalósítás

A részletes megvalósítás szakaszban lépésről lépésre mutatjuk be, hogyan épül fel a teljes rendszer a backendtől a frontendig, milyen technológiákat, mintákat és eszközöket alkalmazunk, valamint mennyi időt vesz igénybe az egyes alrendszerek kidolgozása.

### 7.1 Backend fejlesztés (Laravel, REST API)

A backend a projekt gerince: a Laravel keretrendszer gyorsítja a fejlesztést, beépített támogatást nyújt a migrációkhoz, validációhoz, hitelesítéshez és üzenetkezeléshez.

Első lépések A fejlesztést egy kész Laravel projekt előkészítésével indítjuk:

* Composerrel létrehozzuk a vázat (Laravel 10.x).
* .env fájlba beállítjuk az adatbázis-kapcsolatot, cache- és queue-driverként Redis-t választunk.
* Telepítjük a JWT hitelesítéshez szükséges csomagot (tymon/jwt-auth) és konfiguráljuk az auth.php-ban. Ez a fázis 2 fejlesztői napot igényel.

Modellek és kapcsolatok A következő entitásokat hozzuk létre Eloquent-modellekként:

* User (tárolja a recepciósok, szerelők és adminok adatait)
* Vehicle (rendszám, gyártó, modell, évjárat)
* Service (szolgáltatástípus, időtartam, díj)
* Booking (dátum, időintervallum, státusz)
* Alert (értesítés típusa, ütemezés, státusz)
* CalendarEvent (ICS-exportnál használt adatok)

A Booking és Service között N:M kapcsolatot pivot-táblával oldjuk meg (booking\_service). Minden migrációt migrációs fájlokban definiálunk, idegen kulcsokkal és indexekkel a teljesítmény optimalizálásához. Ez 3 nap alatt készül el.

Üzleti logika réteg A Controller–Service–Repository mintát választjuk:

* A Controller csak a HTTP-kérések kezeléséért és válaszok előállításáért felelős.
* A Service-osztályok tartalmazzák az üzleti szabályokat (például időpont-ütközés-ellenőrzés, összetett szolgáltatáscsomagok kezelése).
* A Repository réteg közvetlenül az Eloquent-modelleken keresztül végzi az adatbázis-műveleteket, így a későbbi adattároló-váltás (például NoSQL) egyszerű marad.

A tranzakciókezelést a Laravel DB::transaction() API-jával oldjuk meg, hogy több táblába írásnál (foglalás, alert, calendar\_event) vagy minden mentés vagy semmi alapon működjön.

Ez a szakasz 5 nap alatt valósul meg, beleértve az unit tesztek írását (PHPUnit) minden Service- és Repository-osztályra.

API végpontok és validáció A REST API útvonalakat routes/api.php-ben szervezzük egységes v1 prefix-szel és auth:api middleware-rel. A kérések validálását Laravel FormRequest osztályok végzik, ahol minden mezőre részletes szabályokat szabunk (pl. rendszám regex, dátum jövőbeli időpont). Hibakezeléskor egységes JSON-választ adunk 400–422–500 státuszkódokkal, egyedi hibaüzenetekkel.

A végpontok dokumentálását automatikus OpenAPI-generátorral (pl. darkaonline/l5-swagger) támogatjuk, így minden új Controller-módszer automatikusan megjelenik a Swagger-felületen.

Összesen 4 nap megy el a végpontok teljes implementációjára, dokumentálására és tesztelésére.

Aszinkron feladatok és értesítések Az SMS- és e-mail-értesítések, valamint a naptárszinkronizáció külső API-hívásai hosszú futású műveletek. Ezeket Laravel Queue–ban regisztrált Job-okban valósítjuk meg, granuláris retry-stratégiával (exponenciális backoff). A Queue futtatását Redis + Horizon monitorozza. A CalendarEvent-ek generálása és ICS-export is queue-olt feladatban történik.

Ez a modul 3 napot vesz igénybe, beleértve a sandbox-os külső API tesztelését és a fallback mechanizmusok kidolgozását.

CI/CD és kódminőség Minden merge után GitHub Actions lefuttatja a lintelést (PHP-CS-Fixer), a statikus elemzést (PHPStan), a unit és integrációs teszteket. Csak zöld pipeline esetén deploy-ol a tesztkörnyezetbe. Ez az infrastruktúra felállítása 2 nap munkát igényel.

**Backend összesen:** 19 fejlesztői nap

### 7.2 Frontend fejlesztés (React)

A frontend a felhasználói interakciók helyszíne. Gyors, reszponzív és akadálymentes felületet építünk Reacttel, TypeScript nélkül, de szigorú PropTypes-szal.

Kezdőállapot és eszközök

* Create React App használata testreszabott Webpack-konfigurációval
* ESLint + Prettier előre definiált szabályok
* Axios HTTP-kliens, alapértelmezett interceptors-beállítások JWT-válasz fejlécek kezeléséhez
* React Router Dom útvonalkezelés, védett route-ok AuthGuard komponenssel

Ez 1 nap alatt előkészíthető.

Alapvető oldalak és komponensek A projekt szerkezete moduláris:

* /pages – login, register, dashboard, booking, calendar, profile
* /components – UI-elemek (űrlapmezők, modalok, gombok)
* /hooks – egyedi hookok az API-hívások, formakezelés és auth állapot kezelésére
* /services – API-kapszulák Axios-alapból

Minden oldal egy layout-komponenst használ, amely sablonos fejlécet és navigációs menüt tartalmaz. A Dashboard valós idejű foglalási összegzést jelenít meg kártyákon, míg a Booking oldal wizard-stílusban végigvezeti a felhasználót a szolgáltatás-, időpont- és járműválasztáson.

A komponensek fejlesztése és egységtesztelése (Jest + React Testing Library) 8 napot igényel.

Stílus és reszponzivitás Styled-components segítségével témakezelést építünk:

* Fő témavariáns: világos és sötét mód
* Mobil első megközelítés breakpoint-okkal (320px, 768px, 1024px)
* WCAG 2.1 AA kontraszt és billentyűzetes navigáció tesztelése a Storybook keretében

Ez 4 nap, beleértve a Storybook komponens-katalógus felépítését.

Űrlapkezelés és validáció A Formik + Yup párost alkalmazzuk, ahol minden mezőhöz komplex feltételrendszer kapcsolódik (rendszám regex, szolgáltatások kiválasztása kötelező, dátum jövőbeni és munkaidőn belüli). A formik-wizard kiegészítővel többoldalas űrlapot építünk, ahol a felhasználó előre-hátra is navigálhat anélkül, hogy elveszne az adata.

Ez 3 nap, beleértve a hibakezelő üzenetek lokalizálását.

Naptárnézet A teljes naptármegjelenítéshez a FullCalendar komponensét illesztjük be, dübörgő adatfrissítéssel:

* WebSocket-en érkező booking-eventek azonnali frissítés
* Drag-and-drop időpontmódosítás
* ICS-export gomb

A WebSocket-létrejöttet a Socket.io kliens oldali csomaggal valósítjuk meg, amely a backend Broadcast eseményeit kezeli. Ez 5 nap, beleértve az e2e teszteket Cypress-szel.

**Frontend összesen:** 21 fejlesztői nap

### 7.3 Adatbázis-implementáció (MySQL)

A MySQL adatbázis a perzisztencia bázisa, normalizált entitásmodellel és teljesítményre hangolt indexeléssel.

Migrációk és séma Minden tábla migrációját Laravel Migration osztályokkal hozzuk létre:

* users, password\_resets, vehicles
* services, bookings, booking\_service
* alerts, calendar\_events

Az idegen kulcsokra ON DELETE CASCADE beállítást alkalmazunk, hogy a törlések ne hagyjanak elhagyott rekordokat. Kiegészítésként fulltext indexet helyezünk el a services.leírás mezőn, hogy adminisztrációs oldalról gyors keresést tegyünk lehetővé.

Ez 2 nap.

Seedek és factory-k Fejlesztési és tesztelési célra minden entitáshoz létrehozunk factory-kat fakerrel, és seed-szkripteket:

* 50 felhasználó (különböző szerepkörökben)
* 200 jármű, hozzárendelve a felhasználókhoz
* 20 szolgáltatás
* 500 random foglalás több hónapos időintervallumban

Ezzel a tesztadatok előállítása és a teljes kör integrációs tesztek futtatása automatizáltan megvalósítható. 1 nap az implementáció, 1 nap a validálás és a seed-ek karbantartása.

Optimalizálás A leggyakoribb lekérdezéseknél (szabad időpontok, naptárfeed) composite indexeket (dátum + státusz) alkalmazunk, valamint a readonly replikára irányítjuk át az olvasó műveleteket. A cache layer Redis-ben tárolja a services és vehicles listákat 5 perces TTL-lel, így a frontenden a válaszidő 20–50 ms körül tartható.

Ez 1 nap.

**Adatbázis összesen:** 5 fejlesztői nap

### 7.4 Tiszta kód irányelvek alkalmazása

A tiszta kód (Clean Code) koncepcióját már a tervezéstől kezdve követjük. Folyamatosan érvényesítjük a PSR-12 kódsztílust, komponensszintű Single Responsibility Principle-t (SRP) és Dependency Injection-t.

Kódszerkezet és névkonvenciók

* Mappastruktúra: App\Http\Controllers\Api\V1, App\Services, App\Repositories
* Osztály- és metódusnevek self-dokumentálóak, egyértelmű felelősségi körrel
* Konfliktusok elkerüléséhez csatlakozó névtereket alkalmazunk

Automatizált ellenőrzések

* PHP-CS-Fixer és ESLint minden push-nál
* SonarQube Quality Gate: coverage ≥ 80%, bugs ≤ 5, vulnerabilities ≤ 0
* Pull requestenként legalább egy peer code review

Dokumentáció és kommentek Minden publikus osztály és metódus PHPDoc-kommentet kap, ahol a paraméterek és visszatérési értékek típusát, valamint a kivételek esetleges dobását is dokumentáljuk. A frontend komponensek mellé JSDoc-kommenteket írunk, és a Storybook markdown-blockjaiban mutatjuk be a proptypes-okat.

Folyamatos refaktorálás A sprint retrospective események után kijelölt technikai adósságokat (tech debt) backlogban tartjuk, és minden sprintben legalább egy napot szánunk ezek felszámolására. Ez biztosítja, hogy a kódbázis hosszú távon is tiszta és karbantartható maradjon.

**Összesített implementációs idő**: 45 fejlesztői nap, amely sprintenként elosztva 6 sprint alatt valósul meg. A tiszta kódra, tesztekre és dokumentációra fordított idő garantálja, hogy a végeredmény megbízható, átlátható és könnyen bővíthető alkalmazás legyen.

## 8. Tesztelés

A tesztelés célja, hogy az Autószerviz Időpontfoglaló Rendszer minden komponense stabilan, a követelményeknek megfelelően működjön, és a kódminőség folyamatosan javuljon. A következőkben részletesen bemutatjuk a tesztelési stratégiát, a felhasznált eszközöket, a tesztfajták lebontását, valamint a teszteredmények gyűjtésének és dokumentálásának folyamatát.

### 8.1 Unit tesztek

A unit tesztek keretében az alkalmazás legkisebb egységeit, azaz egy-egy osztály vagy függvény működését ellenőrizzük izolált környezetben. A backend oldalon Laravelben PHPUnit tesztekkel fedjük le minden Service- és Repository-osztály üzleti logikáját, validációs szabályait és hibakezelését. Minden teszti szcenárió mock objektumokra épül, hogy a külső függőségek (pl. adatbázis, külső API-k) ne befolyásolják a tesztek determinisztikus lefutását. A cél, hogy a kritikus komponenseknél legalább 80–90 százalékos kódlefedettséget (coverage) érjünk el, és minden új commit után a CI pipeline automatikusan lefuttassa ezeket a teszteket.

A frontend unit teszteket Jest és React Testing Library párosával valósítjuk meg. Minden egyes React-komponenst külön tesztfájl kísér, ahol a renderelés, a propok helyes kezelése és az eseménykezelők működése kerül ellenőrzésre. A tesztkódokban snapshot teszteket alkalmazunk, hogy a komponensek markup-ja változáskor könnyen nyomon követhető legyen. A unit tesztek átlagosan 100–200 ms alatt futnak le, így minden kódmódosítás gyors visszajelzést biztosít, és támogatja az agilis fejlesztési ciklust.

### 8.2 Integrációs tesztek

Az integrációs tesztek során a rendszer részeit – például egy Controller, a hozzá tartozó Service és a valódi adatbázis-kapcsolat – együtt futtatjuk, hogy biztosítsuk az összekapcsolódások hibamentes működését. A backend integrációs tesztjeinkben Laravel in-memory SQLite adatbázist használunk, és minden futtatás előtt migrációkat és seed adatokat töltünk be. Így teszteljük a REST API-k endpontjain keresztüli adatfelvételt, módosítást és törlést, az autentikációs folyamatokat, valamint a tranzakciókezelést.

Frontend oldalon az integrációs tesztek kiterjednek arra, hogy a React-alkalmazás valódi HTTP-hívásokat intézzen a tesztkörnyezetben futó mockolt vagy staging backendre. Így ellenőrizzük, hogy az API-service modulok helyesen továbbítják a JSON-adatokat, a hook-ok megfelelően frissítik a komponensállapotot, és a felhasználói események (kitöltött űrlap, gombnyomás) eredménye ténylegesen megjelennek a felületen. Minden integrációs teszt 1–2 másodperc alatt lefut, és a CI pipeline részeként napi több alkalommal is ellenőrzi a rendszer működését.

### 8.3 End-to-end tesztek

Az end-to-end (E2E) tesztek a felhasználó szemszögéből futtatnak teljes munkafolyamatokat a böngészőben. Cypress-szel automatizáljuk a regisztráció, belépés, szolgáltatásválasztás, időpontfoglalás, módosítás és értesítés-visszaigazolás lépéseit. A tesztek valós környezetet modelleznek: a staging szerveren futó alkalmazással kommunikálnak, és a valós SMS- vagy e-mail-szolgáltató sandbox API-ját használják. A forgatókönyvek közé tartozik a párhuzamos foglalási kísérlet, a naptár drag-and-drop módosítása és az ismétlődő emlékeztetők ellenőrzése.

Az E2E teszteket parancssori lefuttatásra és grafikus felületen való interaktív debuggolásra egyaránt optimalizáljuk. A CI környezetben headless módban futnak, és minden teszt végén képernyőmentést, videót, valamint részletes parancssori logokat generálnak. Egy teljes E2E tesztcsomag lefutása átlagosan 10–15 percet vesz igénybe, ezért a pipeline-ban párhuzamos szálakon skálázzuk a végrehajtást, hogy a fejlesztők minél gyorsabb visszajelzést kapjanak.

### 8.4 Teszteredmények dokumentálása

A tesztek eredményeinek átlátható dokumentálása kulcsfontosságú a minőségbiztosítás és a projekt transzparenciája szempontjából. A PHPUnit és Jest futtatás JUnit-kompatibilis XML-jelentéseket állít elő, amelyeket a GitHub Actions Artifacts-ként töltenek fel és Grafana dashboardra át is küldhetnek. A Cypress HTML- és JSON-riportereket generál, amiket a staging környezet „Test Reports” aloldalán érhetővé teszünk.

A SonarQube folyamatosan integrálja a kódlefedettségi és statikus analízis eredményeket, így a technikai vezetők és QA mérnökök egy közös felületen követhetik a hibaszám alakulását, a code smell mutatókat és a biztonsági figyelmeztetéseket. Minden sprint végén egy tesztelési összefoglaló dokumentum készül, amely részletesen tartalmazza az elvégzett tesztfajtákat, a lefedettségi mutatókat, a kritikus hibákat, azok státuszát, és a visszajavítás idejét. Ez az összefoglaló része a végleges projektcsomagnak, amelyet a vizsgaelőtt 14 nappal kell benyújtani.

## 9. Telepítés és üzemeltetés

### 9.1 Deployment pipeline

A rendszer folyamatos telepítési folyamata GitHub Actions alapú CI/CD pipeline-ban fut, ami minden commit és pull request esetén automatikusan elindul. Az első szakasz a forráskód építése és lintelése: itt a backend PHP-CS-Fixer, PHPStan, a frontend pedig ESLint, Prettier és TypeScript ellenőrzésen esik át. Ha minden statikus és unit teszt zölden teljesül, a Docker-képfájlok elkészülnek, majd privát konténerregisztrációs szolgáltatóba (pl. GitHub Packages vagy AWS ECR) kerülnek tolóparancsokkal.

A második fázis a staging környezetbe való deploy, ahol a Helm chartok vagy Kubernetes YAML-ek alkalmazása történik meg Kubernetes API-n keresztül. A pipeline itt végrehajtja az adatbázis-migrációkat helm upgrade --install parancson át, és beállítja az environment-specific ConfigMap-eken, Secret-eken keresztüli konfigurációt. A staging deploy után automatikus integrációs és end-to-end tesztek (Cypress) futnak, és csak sikeres futás esetén válik elérhetővé a Prod jóváhagyás.

A harmadik lépcső a kézi vagy automatikus jóváhagyás utáni éles deploy: blue–green vagy canary stratégiával telepítünk, így az új verzió párhuzamosan futhat a régivel, majd fokozatosan cseréljük le a forgalmat. A pipeline tartalmaz rollback lépéseket is, amelyek a korábbi stabil verzióra állítanak vissza egyetlen parancs segítségével. Minden deploy során részletes naplósorokat gyűjtünk, és gyors visszakapcsolást teszünk lehetővé, ha az új verzió egészségellenőrzése (readiness probe) hibát jelez.

### 9.2 Környezetek kezelése

A fejlesztési, teszt- és éles környezetek elkülönítése biztosítja, hogy a csapat párhuzamosan dolgozhasson anélkül, hogy egymás munkáját zavarják. A fejlesztői környezet lokálisan Docker Compose-szal futtatható, tetszőleges adatbázis- és cache-mockokkal, így gyors indítást és hibakeresést tesz lehetővé. A központi tesztkörnyezet Kubernetes klaszterben helyezkedik el, ahol minden sprint végén frissítjük az épp elkészült verzióval, és az integrációs tesztek mellett a load tesztek is itt futnak.

Az éles környezetet több availability zone-ban üzemeltetjük, dedikált VPC-ben és önálló adatbázispéldányokon, read-replikákkal. A konfigurációk kezeléséhez Kubernetes ConfigMap-eken és Secrets-en kívül HashiCorp Vault szolgál, amely dinamikus titkoskulcs-generálást és automatikus elérési jogkezelést kínál. Minden környezethez eltérő resource quota és autoscaling szabályok tartoznak, így a teszt- és staging környezetek költségkímélőn méreteződnek, míg az éles klaszter a 70 % feletti terhelés esetén is képes új podokat indítani.

A környezetekhez tartozó hozzáféréseket IAM-szabályok korlátozzák: a fejlesztők csak fejlesztői namespace-ekhez, a QA a teszt-namespace-ekhez, az üzemeltetők pedig mindháromhoz felügyeleti joggal férnek hozzá. A pipeline-ba épített “approve” lépések biztosítják, hogy egy adott deploy csak a megfelelő felelős jóváhagyása után kerüljön a következő környezetbe. Ezzel biztosítjuk a környezetek közötti szigorú szeparációt és auditálhatóságot.

### 9.3 Frissítések és karbantartás

Az éles rendszer frissítéseit havi szinten ütemezzük, karbantartási ablakokat kijelölve hétköznap késő esti időszakra, amikor a forgalom a legalacsonyabb. A frissítések magukban foglalják az alkalmazásversiók cseréjét, az adatbázis-migrációk futtatását és a konténerképek frissítését. Minden adatbázis-migrációs lépést rollback szkript kísér, és az új verzió élesítése előtt először tesztadatbázison validáljuk a migrációk helyességét.

A rendszer napi biztonsági mentést készít teljes adatbázisról és konfigurációs állományokról, míg óránként az egyes kritikus adattáblákról tranzakciós naplófájl-mentések (binlog) készülnek. A mentéseket 30 napig tartjuk meg, és legalább havonta egyszer szimulált visszaállítást végzünk, hogy megbizonyosodjunk a helyreállítási folyamat működéséről. A mentések felhőalapú tárolókba (S3 kompatibilis szolgáltatás) kerülnek, redundáns régióban tárolva az adatsérülés kockázatának minimalizálására.

A folyamatos üzemeltetést monitoring és alerting eszközök (Prometheus, Grafana, ELK) támogatják. A szolgáltatások egészségét liveness és readiness probe-ok figyelik, a kulcsmetrikák (válaszidő, hibaarány, CPU-, memóriahasználat) folyamatosan gyűlnek, és küszöbérték átlépésekor az on-call üzemeltető értesítést kap SMS-ben vagy Slack-csatornán. A hiba esetén előre definiált runbook vezeti végig a csapatot a vizsgálat és hibaelhárítás lépésein, az incidensek dokumentálása pedig a post-mortem folyamat része lesz a jövőbeni problémák megelőzése érdekében.

## 10. Dokumentáció és mellékletek

### 10.1 Felhasználói kézikönyv (ügyfél és adminisztrátor)

A felhasználói kézikönyv első fejezete rövid áttekintést ad a rendszer céljáról és a támogatott felhasználói szerepkörökről. Itt tisztázzuk, kik tartoznak az ügyfél és az adminisztrátor csoportba, és milyen jogosultságokkal rendelkeznek. Minden szerepkörhöz külön bevezető ismerteti a belépés módját, a főképernyők elérését, valamint az ügyféloldali és adminisztrációs feladatok alaplépéseit.

A dokumentum második részében képernyőképekkel ellátott, lépésről lépésre követhető útmutatás található. Az ügyfél fejezet a regisztrációtól indul: bemutatja a regisztrációs űrlap mezőit, a jelszó- és e-mail-validáció működését, majd végigvezeti a mobilbarát foglalási folyamat minden lépésén (szolgáltatás kiválasztása, rendszer lekérdezése, adatok kitöltése, foglalás véglegesítése). A kézikönyv külön kitér a foglalás módosítására, törlésére és a naptárszinkronizáció indítására, bemutatva az ICS-export gomb használatát és a Google/Exchange naptárhozzáadás lépéseit.

Az adminisztrátoroknak szóló rész részletesen ismerteti a recepciós és szerelői munkafázisokat. Leírja, hogyan tekinthetők meg a beérkező foglalások listái, hogyan kezelhetők a szolgáltatáskatalógus elemei (új szolgáltatás hozzáadása, díj- és időtartam-módosítás), valamint miként lehet gyorsan reagálni a párhuzamos foglalási ütközésekre. A fejezet végén egy „gyakori hibák és megoldásaik” gyűjtemény segíti a felhasználót a legjellemzőbb problémák gyors orvoslásában.

### 10.2 Fejlesztői dokumentáció

A fejlesztői dokumentáció a rendszer architektúrájának és kódstruktúrájának átfogó leírását tartalmazza. A bevezető fejezet bemutatja a háromrétegű szerkezetet: Laravel alapú REST API backend, React frontend kliens és MySQL adatbázis. Minden réteghez külön áll rendelkezésre UML-komponens-, osztály- és komponensdiagram, amelyek vizuálisan ábrázolják az egymásra épülő modulokat és azok interfészeit.

A középső fejezet részletezi a fejlesztési környezet felállítását: Git-clone lépések, környezeti változók beállítása .env fájlban, Docker–Compose és Kubernetes konfigurációk, valamint a CI/CD pipeline (GitHub Actions) lépései. Példák és parancssori kódrészletek mutatják be a migrációk futtatását, a tesztek elindítását és az alkalmazás staging környezetbe telepítését. A dokumentum kitér a SonarQube-integrációra is, ismerteti a Quality Gate definícióit és a kódminőségi metrikák figyeléséhez szükséges beállításokat.

A harmadik rész a forráskód konvencióit és patternjeit taglalja. Leírja a Controller–Service–Repository rétegek felelősségét, a Dependency Injection használatát, a FormRequest validációs osztályok kialakítását és a DTO-k (Data Transfer Object) koncepcióját. A frontendnél a moduláris mappastruktúrát, a React Hook-ok és context-ek állapotkezelését, a styled-components témarendszerét, valamint a routing- és auth-guard mechanizmusokat dokumentálja. Végül az API végpontokat OpenAPI-specifikációban is mellékeli, így a swagger.json minden egyes mezőjét értelmezhetővé teszi.

### 10.3 Tesztelési dokumentáció és eredmények

A tesztelési dokumentáció áttekintést nyújt a tesztstratégiáról, a teszteszközökről és a lefedettségi követelményekről. Az első fejezetben összefoglaljuk a unit tesztek célját, a PHPUnit és Jest konfigurációját, valamint a mock- és stub-alapú izolált tesztelés szabályait. Részletes példa mutatja be, hogyan írjunk Service-osztályhoz unit tesztet Laravelben és komponens tesztet Reactben, beleértve a snapshot-ellenőrzést.

A következő rész az integrációs és end-to-end tesztek lebonyolítását dokumentálja. A backend integrációs teszteknél bemutatjuk az in-memory SQLite adatbázis használatát, migrációk automatikus betöltését és az autentikációs middleware tesztelését. Az E2E tesztekhez a Cypress konfigurációs fájlokat, parancssori futtatási opciókat és a parallelizálás beállításait mellékeljük. A dokumentumban külön fejezet foglalkozik a headless és GUI módban való futtatással, valamint a képernyőmentések és videók automatikus gyűjtésének módjával.

A teszteredmények összegyűjtésére szolgáló reportok formátumát és tárolási helyét is rögzítjük. Megmutatjuk, hogyan exportálódnak a JUnit XML fájlok, hogyan épülnek be a GitHub Actions Artifacts-be, és miként érhetők el a Grafana dashboardon. Szintén részletezzük a SonarQube kódlefedettségi riportjainak értelmezését, a legfontosabb metrikák (coverage, bugs, vulnerabilities) áttekintését és a hibák backlogba kerülésének folyamatát.

### 10.4 GitHub repository és telepítőkészlet

A teljes forráskód elérhető a GitHub-on egy nyilvános repository-ban, amelyben a main ágon található a stabil, kiadásra kész verzió, míg a develop ágon zajlik a napi integráció. A README.md fájl részletesen ismerteti a projekt előfeltételeit (Docker, Kubernetes CLI, PHP 8.x, Node.js 16+), a klónozás és a helyi indítás lépéseit, valamint a Branch Policy és Commit Convention (Conventional Commits) szabályait.

A natív asztali telepítőkészlet Windows és macOS platformokra készült. Windows esetén egy MSI-csomaggal, macOS esetén egy Electron alapú .dmg fájllal szolgálunk. A telepítőkészlet tartalmazza a backend Docker-képet tarballként, a frontend statikus buildet és egy egyszerű „Install.bat” vagy „Install.sh” scriptet, amely az adatbázisok migrációját és a seedek futtatását is elvégzi. Egy részletes telepítési útmutató PDF formátumban mutatja be a script használatát, a környezeti változók beállítását és a gyakori hibaelhárítási lépéseket.

### 10.5 Adatbázis dokumentáció és export fájlok

Az adatbázis-modelldiagram egy A3-as oldalon elhelyezett, harmadik normálforma szerinti ERD, amely bemutatja az entitásokat, attribútumaikat, kulcsaikat és a táblák közötti relációkat. A diagramot draw.io formátumban és exportként PNG és PDF változatban mellékeljük, így könnyen módosítható és nyomtatható.

Az exportált adatbázis-dump egy dump.sql fájl, amely tartalmazza az összes tábla szerkezetét és az alap seed-adatokat (például alapértelmezett szolgáltatáskatalógus, tesztfelhasználók). A dokumentum részletezi, hogyan importálható be ez a fájl MySQL-be (mysql -u user -p database < dump.sql), és milyen lépéseket kell tenni a jogszabályi megfelelés érdekében (adatminimalizálás, GDPR-procedúrák). A dump mellé csatolunk egy rövid PowerShell- és Bash-szkriptet is, ami automatikusan elvégzi a mentést és visszaállítást.

# **Üzleti kontextus és elemzések**

### Piaci helyzetkép az autószerviz-időpontfoglaló megoldások terén

Az autószerviz-piac hazai és nemzetközi szinten egyaránt átalakulóban van: a tradicionális, telefonos és papíralapú ügyfélkezelés helyét egyre inkább a digitális platformok átütő ereje váltja fel. Magyarországon a felnőtt lakosság közel 80 %-a naponta legalább egyszer használ okostelefont, és több mint 60 %-uk számítógépet is munkája vagy magánélete során【forrás szükséges】. Ezzel párhuzamosan az autótulajdonosok körében egyre inkább elvárássá válik, hogy a szervizelés a lehető legkisebb ügyintézéssel, gyorsan és pontosan történjen. A COVID-19 járvány felgyorsította az érintésmentes szolgáltatások iránti igényt: a kliensek 0–24 elérhető foglalási lehetőséget keresnek, és hajlandóak prémium díjat is fizetni a gördülékeny, felhasználóbarát online rendszerekért.

A piacon jelenleg több nemzetközi és hazai szereplő kínál általános időpontfoglaló megoldásokat (pl. Calendly, Google Calendar), de ezek legtöbbször nem illeszkednek az autószervizek speciális folyamataihoz (szerviztípusok, munkafázisok, járműadatok, alkatrészigények). A szegmentált, szerviz-specifikus platformok – bár célzottabb funkcionalitást kínálnak – jellemzően drágább licencekhez vagy bonyolult bevezetési folyamathoz kötődnek, így a kis- és középvállalkozások számára elérhetetlenek maradnak. A piaci résztvevőknek komoly lehetőség a könnyen bevezethető, „out-of-the-box” megoldások beárazása és csomagajánlatai révén, amelyek egyszerre kínálnak webes és mobilélményt.

Jelenleg a hazai szervizek jelentős része (mintegy 70 %) még mindig manuális módon bonyolítja a foglalásokat, ami magas adminisztrációs költségeket és rengeteg emberi hibalehetőséget rejt magában. Ezzel szemben a digitális átállás költségei – domain, tárhely, fejlesztés és karbantartás – 2–3 millió forint körül mozognak, de a beruházás már az első évben képes 20–25 %-os üzemeltetési megtakarítást hozni a papíralapú rendszerrel szemben, köszönhetően a munkaerő-igény csökkenésének és a párhuzamos visszaigazolások minimalizálásának.

### Célcsoport- és persona-leírások

A rendszer elsődleges célcsoportját az autótulajdonos ügyfelek, valamint a foglalásokat kezelő recepciósok és szerelők alkotják. Az alábbi personaszemléletek segítenek mélyen megérteni a felhasználók motivációit, fájdalompontjait és napi rutinját.

Persona 1 – “Sára, a rohanó dolgozó” Sára Budapesten élő középvezető, akinek munkahelye és családja mellett kevés ideje marad az autó karbantartására. A munkahelyén gyakran egyeztet konferenciahívásokat, ezért akkor szeret foglalni, amikor a legkevesebb a határidő. Számára létfontosságú az azonnali visszaigazolás, az egyszerűség (például korábbi adatok előhívása) és a rugalmas időpontválasztás, akár munkaszüneti napokra is. Fájdalompontja, hogy korábban telefonon a vonal gyakran foglalt, vagy a recepciós hiányos információkat rögzített, ami miatt többször kellett újrahívnia a szervizt.

Persona 2 – “Gábor, a szervizvezető” Gábor egy 12 főt foglalkoztató családi vállalkozás szervizvezetője, aki napi szinten kezeli a járműbeosztást és a munkaerő-tervezést. Munkáját papírlapokon és Excel-táblázatokban vezeti, amit hetente egyetlen alkalommal frissít. Ez okozza a legtöbb párhuzamos foglalási ütközést és a túlterheltség érzését a csapatban. Gábor fő hajtóereje, hogy minimalizálja a kitöltendő űrlapokat, csökkentse a hibás beosztásokat, és valós idejű rálátással rendelkezzen arra, milyen erőforrásokat kell átcsoportosítania.

Persona 3 – “Anna, a recepciós” Anna egy multi-üzemű szervizlánc recepciósa, akinek feladata megszűrni a bejövő hívásokat, új foglalásokat rögzíteni, és emlékeztetőket küldeni. A jelenlegi rendszerben naponta átlagosan 50 SMS-t vagy e-mailt küld manuálisan, ebből 5–7 alkalommal téves időpontot rögzít, vagy elfelejti visszaigazolni. Anna számára a legnagyobb segítséget egy automatikus értesítési modul és a kézi hibalehetőségek radikális csökkentése jelentené, mert így csupán a kivételes esetekkel kellene foglalkoznia.

E három persona összefonódó igényei és elvárásai képezik a rendszer funkcionális mínimumát: gyors ügyfélinterakció, valós idejű kapacitásellenőrzés és automatikus kommunikáció.

### Versenytárs-elemzés és SWOT

A szerviz-időpontfoglaló piac szereplői között megkülönböztethetünk általános online foglalóplatformokat, vertikális iparági megoldásokat és belső fejlesztésű egyszeri rendszereket. Az általános eszközöket a nagyobb rugalmasság és alacsonyabb licencár jellemzi, ám hiányzik belőlük a járműspecifikus logika és a naptárszinkronizációs API-k beépítése. A vertikális megoldások (például néhány külföldi szervizlánc saját platformja) drága előfizetéssel és komplex üzemeltetési igénnyel bírnak, így kkv-knak ritkán megfizethetők. A belső fejlesztések egyedi igényekre szabottak, de fenntartási és bővítési költségeik hosszú távon kiszámíthatatlanok.

| **Terület** | **Erősségek** | | **Gyengeségek** |
| --- | --- | --- | --- |
| Ügyféloldali élmény | Mobil- és webbarát, egyszerű wizard-folyamat, azonnali visszaigazolás | | Lokális SMS-szolgáltatók díjszabása, felhasználói onboarding nehézsége |
| Műszaki architektúra | REST API, konténerizálás, Kubernetes-alapú skálázás | | DevOps-kompetencia hiánya kisebb cégeknél, kezdeti üzembe helyezés nehézsége |
| Funkcionalitás | Szolgáltatáskatalógus, járműnyilvántartás, naptárszinkronizáció | | Speciális igények (alkatrészlogisztika, garanciális jogosultságok) hiánya |
| Ár és licencelés | Versenyképes előfizetési díj, rugalmasság | | Költségérzet magas, ha nincs azonnali haszonérzet |
| **Lehetőségek** | | **Veszélyek** | |
| KKV-k digitális átállásának finanszírozási programjai | | Piaci belépési korlát hiánya miatt új szereplők gyors megjelenése | |
| Integráció autógyártói vagy alkatrész-ellátói rendszerekkel | | Adatvédelmi szabályozások szigorítása, GDPR-mulasztások miatti bírságok | |
| Üzleti intelligencia modul: forgalmi és ügyfélviselkedési elemzések | | Technológiai elavulás, elavult keretrendszerek rövid távú támogatása | |

A versenytárs-elemzés alapján világos, hogy a legnagyobb értéket az ügyfélbarát, járműspecifikus modulok, a valós idejű szinkronizáció és a versenyképes árszabás együttesen teremtik meg. A SWOT-táblázat arra mutat rá, hogy a rendszer számára a belépési pont éppen a kkv-szegmens digitalizációs igényeiben rejlik, ugyanakkor a DevOps-támogatás és a GDPR-portfólió erősítésével mérsékelni kell a belépési akadályokat és a szabályozási kockázatokat.

### Várható megtérülés (ROI) és üzleti előnyök

A beruházás megtérülésének modellezéséhez elsőként feltételezzük, hogy egy közepes méretű (évi 3000 foglalás) autószerviznél a projekt teljes fejlesztési és bevezetési költsége – beleértve a licenc, üzemeltetés és képzési díjakat – 7 500 000 forint. Az éves szintű költségcsökkenés a papíralapú, telefonos ügyintézéshez képest a munkaerő- és adminisztrációs kiadások terén közel 2 000 000 forint, emellett az automatizált emlékeztetők révén 10 %-kal csökken a lemondott vagy elfelejtett időpontok száma, ami 300 hiányzó foglalásból 30 plusz tényleges kiszolgálást jelent.

Amennyiben egy átlagos szolgáltatás árát 25 000 forintban határozzuk meg, az éves plusz bevétel csupán a csökkentett lemondások miatt 30 × 25 000 = 750 000 forint, míg az ügyfélmegtartási mutató 5 %-os javulása további 150 foglalást eredményez (3 750 000 forint). Így az éves többletbevétel összesen mintegy 4 500 000 forint, ami önmagában 60 %-os megtérülést jelent. A valós megtérülés azonban magasabb, ha hozzávesszük a skálázhatóságból fakadó piaci bővülést és a jobb erőforrás-kihasználást, ami további 1–2 millió forintos megtakarítást vagy bevételnövekedést jelenthet.

ROI = (Éves nettó haszon – Éves költségcsökkenés) / Bevezetési költség ROI = (4 500 000 + 2 000 000 – 0) / 7 500 000 ≈ 0,87 → 87 %

Az önerőből történő fejlesztés esetén a megtérülési időszak 14–18 hónapra tehető, míg szolgáltatásként igénybevett modellnél a havi előfizetési díj (200 000 forint/hó) mellett a belépéshez nem kell egyszerre nagy tőkebefektetés, és a szolgáltató garantálja a folyamatos frissítést, karbantartást.

A legfontosabb üzleti előnyök összefoglalva:

* Adminisztrációs költségek jelentős csökkenése
* Csökkenő lemondási arány és magasabb foglalási kihasználtság
* Ügyfél-elégedettség és lojalitás erősödése
* Valós idejű adatgyűjtés: beavatkozási lehetőségek optimalizálása
* Versenyelőny a kkv-szegmensben, amely most kezdi a digitális átállást

Ezek a tényezők együtt biztosítják, hogy a rendszer ne csak rövid távon, hanem hosszú távon is üzleti értéket teremtsen, és fenntartható növekedést generáljon a szerviz számára.

# Üzleti kontextus és elemzések

### Piaci helyzetkép az autószerviz-időpontfoglaló megoldások terén

Az autószerviz-piac hazai és nemzetközi szinten egyaránt átalakulóban van: a tradicionális, telefonos és papíralapú ügyfélkezelés helyét egyre inkább a digitális platformok átütő ereje váltja fel. Magyarországon a felnőtt lakosság közel 80 %-a naponta legalább egyszer használ okostelefont, és több mint 60 %-uk számítógépet is munkája vagy magánélete során【forrás szükséges】. Ezzel párhuzamosan az autótulajdonosok körében egyre inkább elvárássá válik, hogy a szervizelés a lehető legkisebb ügyintézéssel, gyorsan és pontosan történjen. A COVID-19 járvány felgyorsította az érintésmentes szolgáltatások iránti igényt: a kliensek 0–24 elérhető foglalási lehetőséget keresnek, és hajlandóak prémium díjat is fizetni a gördülékeny, felhasználóbarát online rendszerekért.

A piacon jelenleg több nemzetközi és hazai szereplő kínál általános időpontfoglaló megoldásokat (pl. Calendly, Google Calendar), de ezek legtöbbször nem illeszkednek az autószervizek speciális folyamataihoz (szerviztípusok, munkafázisok, járműadatok, alkatrészigények). A szegmentált, szerviz-specifikus platformok – bár célzottabb funkcionalitást kínálnak – jellemzően drágább licencekhez vagy bonyolult bevezetési folyamathoz kötődnek, így a kis- és középvállalkozások számára elérhetetlenek maradnak. A piaci résztvevőknek komoly lehetőség a könnyen bevezethető, „out-of-the-box” megoldások beárazása és csomagajánlatai révén, amelyek egyszerre kínálnak webes és mobilélményt.

Jelenleg a hazai szervizek jelentős része (mintegy 70 %) még mindig manuális módon bonyolítja a foglalásokat, ami magas adminisztrációs költségeket és rengeteg emberi hibalehetőséget rejt magában. Ezzel szemben a digitális átállás költségei – domain, tárhely, fejlesztés és karbantartás – 2–3 millió forint körül mozognak, de a beruházás már az első évben képes 20–25 %-os üzemeltetési megtakarítást hozni a papíralapú rendszerrel szemben, köszönhetően a munkaerő-igény csökkenésének és a párhuzamos visszaigazolások minimalizálásának.

### Célcsoport- és persona-leírások

A rendszer elsődleges célcsoportját az autótulajdonos ügyfelek, valamint a foglalásokat kezelő recepciósok és szerelők alkotják. Az alábbi personaszemléletek segítenek mélyen megérteni a felhasználók motivációit, fájdalompontjait és napi rutinját.

Persona 1 – “Sára, a rohanó dolgozó” Sára Budapesten élő középvezető, akinek munkahelye és családja mellett kevés ideje marad az autó karbantartására. A munkahelyén gyakran egyeztet konferenciahívásokat, ezért akkor szeret foglalni, amikor a legkevesebb a határidő. Számára létfontosságú az azonnali visszaigazolás, az egyszerűség (például korábbi adatok előhívása) és a rugalmas időpontválasztás, akár munkaszüneti napokra is. Fájdalompontja, hogy korábban telefonon a vonal gyakran foglalt, vagy a recepciós hiányos információkat rögzített, ami miatt többször kellett újrahívnia a szervizt.

Persona 2 – “Gábor, a szervizvezető” Gábor egy 12 főt foglalkoztató családi vállalkozás szervizvezetője, aki napi szinten kezeli a járműbeosztást és a munkaerő-tervezést. Munkáját papírlapokon és Excel-táblázatokban vezeti, amit hetente egyetlen alkalommal frissít. Ez okozza a legtöbb párhuzamos foglalási ütközést és a túlterheltség érzését a csapatban. Gábor fő hajtóereje, hogy minimalizálja a kitöltendő űrlapokat, csökkentse a hibás beosztásokat, és valós idejű rálátással rendelkezzen arra, milyen erőforrásokat kell átcsoportosítania.

Persona 3 – “Anna, a recepciós” Anna egy multi-üzemű szervizlánc recepciósa, akinek feladata megszűrni a bejövő hívásokat, új foglalásokat rögzíteni, és emlékeztetőket küldeni. A jelenlegi rendszerben naponta átlagosan 50 SMS-t vagy e-mailt küld manuálisan, ebből 5–7 alkalommal téves időpontot rögzít, vagy elfelejti visszaigazolni. Anna számára a legnagyobb segítséget egy automatikus értesítési modul és a kézi hibalehetőségek radikális csökkentése jelentené, mert így csupán a kivételes esetekkel kellene foglalkoznia.

E három persona összefonódó igényei és elvárásai képezik a rendszer funkcionális mínimumát: gyors ügyfélinterakció, valós idejű kapacitásellenőrzés és automatikus kommunikáció.

### Versenytárs-elemzés és SWOT

A szerviz-időpontfoglaló piac szereplői között megkülönböztethetünk általános online foglalóplatformokat, vertikális iparági megoldásokat és belső fejlesztésű egyszeri rendszereket. Az általános eszközöket a nagyobb rugalmasság és alacsonyabb licencár jellemzi, ám hiányzik belőlük a járműspecifikus logika és a naptárszinkronizációs API-k beépítése. A vertikális megoldások (például néhány külföldi szervizlánc saját platformja) drága előfizetéssel és komplex üzemeltetési igénnyel bírnak, így kkv-knak ritkán megfizethetők. A belső fejlesztések egyedi igényekre szabottak, de fenntartási és bővítési költségeik hosszú távon kiszámíthatatlanok.

A versenytárs-elemzés alapján világos, hogy a legnagyobb értéket az ügyfélbarát, járműspecifikus modulok, a valós idejű szinkronizáció és a versenyképes árszabás együttesen teremtik meg. A SWOT-táblázat arra mutat rá, hogy a rendszer számára a belépési pont éppen a kkv-szegmens digitalizációs igényeiben rejlik, ugyanakkor a DevOps-támogatás és a GDPR-portfólió erősítésével mérsékelni kell a belépési akadályokat és a szabályozási kockázatokat.

| **Terület** | **Erősségek** | | **Gyengeségek** |
| --- | --- | --- | --- |
| Ügyféloldali élmény | Mobil- és webbarát, egyszerű wizard-folyamat, azonnali visszaigazolás | | Lokális SMS-szolgáltatók díjszabása, felhasználói onboarding nehézsége |
| Műszaki architektúra | REST API, konténerizálás, Kubernetes-alapú skálázás | | DevOps-kompetencia hiánya kisebb cégeknél, kezdeti üzembe helyezés nehézsége |
| Funkcionalitás | Szolgáltatáskatalógus, járműnyilvántartás, naptárszinkronizáció | | Speciális igények (alkatrészlogisztika, garanciális jogosultságok) hiánya |
| Ár és licencelés | Versenyképes előfizetési díj, rugalmasság | | Költségérzet magas, ha nincs azonnali haszonérzet |
| KKV-k digitális átállásának finanszírozási programjai | | Piaci belépési korlát hiánya miatt új szereplők gyors megjelenése | |
| Integráció autógyártói vagy alkatrész-ellátói rendszerekkel | | Adatvédelmi szabályozások szigorítása, GDPR-mulasztások miatti bírságok | |
| Üzleti intelligencia modul: forgalmi és ügyfélviselkedési elemzések | | Technológiai elavulás, elavult keretrendszerek rövid távú támogatása | |

### Leírás

A hitelesítési folyamat célja, hogy az ügyfél biztonságosan be tudjon lépni a rendszerbe, vagy regisztráljon új fiókot. A hitelesítés után a felhasználó hozzáfér az összes további funkcióhoz – foglaláshoz, járműadat-kezeléshez, naptárszinkronizációhoz –, a rendszer pedig személyre szabott élményt nyújt, és biztosítja, hogy csak jogosult szerepkörök hajthassanak végre adminisztrációs műveleteket.

### Előfeltételek

* A felhasználó elérte a webes vagy mobil belépőoldalt.
* Az internetkapcsolat stabil és HTTPS protokollú.
* A rendszer rendelkezik inicializált OTP- vagy e-mail szolgáltató beállításokkal a jelszó-visszaállításhoz.

### Fő eseményfolyam

1. A felhasználó megnyitja a bejelentkezési űrlapot.
2. A rendszer kéri a felhasználónév (e-mail) és jelszó megadását.
3. A felhasználó kitölti a mezőket és elküldi az űrlapot.
4. A backend FormRequest-ben validálja az e-mail formátumát és a jelszó meglétét.
5. Sikeres validáció után a JWT-auth csomag ellenőrzi a hitelesítő adatokat.
6. A rendszer visszaad egy 200-as státuszkódot és egy rövid élettartamú hozzáférési tokent plusz frissítő tokent.
7. A frontend tárolja a tokent, és átirányítja a felhasználót a fődashboardra.

### Alternatív utak

* 4a. Ha az e-mail formátum érvénytelen, a rendszer 422-es hibaüzenetet ad „Érvénytelen e-mail” szöveggel, és nem folytatja a hitelesítést.
* 5a. Ha a hitelesítő adatok helytelenek, a rendszer visszaadja a 401-es „Hibás felhasználónév vagy jelszó” választ, és a frontend újrapróbálkozást enged.
* 6a. Ha a tokenkiadás során hiba történik (pl. adatbázis-elérés hiba), a rendszer 500-as belső szerverhibát jelez, és a felhasználó értesítést kap.

### Utófeltételek

* A felhasználó állapota be van állítva „Hitelesítve” módra.
* A hozzáférési token érvényesítve lett, és rövid időn belül frissíthető.

UML Use Case Diagram

+-------------------+

| Ügyfél |

+--------+----------+

|

| bejelentkezés / regisztráció

v

+-------------------+

| Időpontfoglaló |

| Rendszer |

+-------------------+

^ ^

| |

regisztrál / \ bejelentkezik

Aktivitásdiagram (Felhasználói hitelesítés)

[Start]

|

v

(Űrlap megjelenítése) --> [Felhasználói adatbevitel]

| |

v v

[Validáció] -- siker --> (JWT kiadása) --> [Bejelentkezett állapot] --> [End]

|

`-- hiba --> [Hibaüzenet megjelenítése] --> [Felhasználó javít] --> vissza a [Felhasználói adatbevitel]

Szekvenciadiagram (Felhasználói bejelentkezés)

Ügyfél Frontend Backend AuthService

| | | |

| POST /login | | |

|-------------->| | |

| | validate() | |

| |-------------->| |

| | | checkCredentials()|

| | |---------------->|

| | |<--------------- token

| |<--------------| |

|<--------------| token, user | |

| Logged in | | |

## 11.2 Use Case UC2: Járműadat-kezelés

### Leírás

Ez a folyamat lehetővé teszi, hogy a felhasználó járművei adatait (rendszám, gyártó, modell, évjárat) egyszerűen rögzítse, módosítsa vagy törölje. A járműadatok összekapcsolódnak a foglalási rekordokkal, így a szervizben a dolgozók mindig az adott autó előzményeit és igényeit látják.

### Előfeltételek

* A felhasználó hitelesített állapotban rendelkezik aktív tokennel.
* Legalább egy üres járműlista vehető fel a fiókban, vagy létezik már jármű.

### Fő eseményfolyam

1. A felhasználó belép a „Járműkezelés” aloldalra.
2. A frontend GET kérésben lekéri a api/v1/vehicles listát.
3. A backend visszaadja a felhasználóhoz tartozó járművek táblázatát.
4. A felhasználó új jármű hozzáadása gombra kattint.
5. A rendszer egy űrlapot jelenít meg kötelező mezőkkel: rendszám, gyártó, modell, évjárat.
6. A felhasználó kitölti és elküldi a formot.
7. A backend FormRequest-ben érvényesíti a rendszám regex szabályát és az évjárat tartományát.
8. Sikeres validáció esetén a vehicles tábla INSERT művelet végrehajtódik, és a rekord visszaküldésre kerül.
9. A frontend frissíti a járműlistát, és visszajelzi a felhasználónak a sikeres felvitelt.

### Alternatív utak

* 7a. Ha a rendszám érvénytelen, a rendszer 422-es választ küld, és a frontend kiemeli a mezőt.
* 4a. Ha a felhasználó választja az egyik meglévő jármű törlését, a frontend DELETE kérést küld, és egy megerősítő modalt mutat.
* 4b. Ha a felhasználó módosítani kíván egy rekordot, a form előtölti a korábbi adatokat, és PUT kérés során ment.

### Utófeltételek

* A vehicles tábla megfelelően módosult (INSERT/UPDATE/DELETE).
* A felhasználó járműlistája naprakész, és kapcsolódó foglalási rekordok konzisztensen kezelhetőek.

+-------------+

| Ügyfél |

+------+------+

|

| kezeli járműveit

v

+-------------+

| Időpont- |

| foglaló |

| Rendszer |

+-------------+

^

|

mutat járműlista

Aktivitásdiagram (Jármű hozzáadása)

[Start]

|

v

(Űrlap megjelenítése)

|

v

[Adatbevitel]

|

v

[Validálás] -- ok --> (INSERT jármű)

| |

`-- hiba --> [Hibaüzenet] --> vissza az [Adatbevitel]

|

v

[Lista frissítése]

|

v

[End]

Szekvenciadiagram (Új jármű rögzítése)

Ügyfél Frontend Backend Database

| | | |

| click Add | | |

|------------>| | |

| | POST /v1/vehicles |

| |--------------------->|

| | | INSERT |

| | |--------->|

| | |<---------|

| |<---------------------|

|<------------| new vehicle | |

## 11.3 Use Case UC3: Időpontfoglalás

### Leírás

Az időpontfoglalás a rendszer központi funkciója. A felhasználó kiválasztja a szolgáltatásokat, a járművet, majd egy dinamikusan frissülő naptárnézetből foglalja le a neki megfelelőt. A backend párhuzamos tranzakciókezeléssel biztosítja, hogy két felhasználó ne foglalhassa le egyszerre ugyanazt az intervallumot.

### Előfeltételek

* A felhasználó hitelesített állapotban van.
* Legalább egy jármű és legalább egy szolgáltatás elérhető a katalógusban.

### Fő eseményfolyam

1. A felhasználó megnyitja a „Foglalás” oldalt.
2. A frontend GET api/v1/services és api/v1/vehicles hívással begyűjti a katalógus- és járműadatokat.
3. A felhasználó kiválasztja a kívánt szolgáltatásokat (többszörös tétel engedélyezett).
4. A komponens kiszámolja az összesített időtartamot, és real-time lekéri a szabad intervallumokat api/v1/bookings/available?duration=X végponton.
5. A naptárnézet AJAX-szal frissül, zölddel kiemelve az elérhető blokkokat.
6. A felhasználó kattint egy blokkra, megadja a kapcsolati adatokat és megerősíti a foglalást.
7. A frontend POST /api/v1/bookings kérést indít a kiválasztott paraméterekkel.
8. A backend tranzakcióban ellenőrzi újra az intervallumot, INSERT-eli a bookings és pivot rekordokat, majd létrehoz egy calendar\_event és egy alert bejegyzést.
9. A sikeres foglalás után azonnali 201-es választ kap a frontend, és a felhasználó képernyőjén megjelenik a visszaigazolás részletekkel.

### Alternatív utak

* 4a. Ha nincs elérhető intervallum az összesített időhöz, a rendszer 204 No Content választ küld, és a frontend felajánl alternatív időpontokat.
* 7a. Ha a POST kérést közben egy másik felhasználó lefoglalta ugyanazt az időt, a backend 409 Conflict hibaüzenetet ad, és a felhasználó visszakerül az intervallum-választó képernyőre.
* 6a. Ha a kapcsolati adatok hiányosak, a frontend megelőzően kliensoldali validációt futtat, és az űrlapot nem engedi elküldeni.

### Utófeltételek

* A foglalás és kapcsolódó adatok konzisztensen rögzítve lettek.
* A felhasználó és a szerviz naptára naprakész, és az értesítési modul készen áll SMS-/e-mail kiküldésre.

UML Use Case Diagram

+-------------+ +------------+

| Ügyfél | | Szerviz |

+-----+-------+ +------+-----+

| |

időpontfoglalás naptárnézet

| |

v v

+-------------------------+ +----------------+

| Foglaló Rendszer | | Naptár Modul |

+-------------------------+ +----------------+

Aktivitásdiagram (Időpontfoglalás)

[Start]

|

v

(Oldal betöltése) --> [Servicék + járművek lekérése]

| |

v v

[Naptár render] <--- [available intervallum lekérése]

|

v

[Blokk kiválasztása]

|

v

[Adatbevitel + validáció]

|

v

[POST foglalás]

|

v

[Tranzakció ellenőrzés]

|

v

[Sikeres visszaigazolás] --> [End]

|

`-- hiba (`409`/`204

# 12. Kódpéldák és konfigurációk

Ebben a fejezetben bemutatjuk a projekt kulcsfontosságú kódrészleteit és konfigurációs mintáit, amelyek a backend architektúra rétegeit, a frontend komponenseket és a teljes CI/CD/infra környezet yaml- és pipeline-beállításait fedik le. Minden kódrészlethez részletes magyarázatot adunk a működésről és integrációról.

## 12.1 Backend kódpéldák és konfigurációk

### 12.1.1 Controller–Service–Repository együttműködés

A Laravel alkalmazásban a Controller csak a HTTP réteget kezeli, a Service réteg tartalmazza az üzleti logikát, a Repository pedig az adatbázis-műveleteket izolálja. Ez a háromrétegű minta támogatja a tiszta kód elvét és egyszerűsíti a unit tesztelést.

// app/Http/Controllers/Api/V1/BookingController.php

namespace App\Http\Controllers\Api\V1;

use App\Http\Requests\StoreBookingRequest;

use App\Services\BookingService;

use Illuminate\Http\JsonResponse;

use App\Http\Controllers\Controller;

class BookingController extends Controller

{

protected BookingService $bookingService;

public function \_\_construct(BookingService $bookingService)

{

$this->bookingService = $bookingService;

}

public function store(StoreBookingRequest $request): JsonResponse

{

$data = $request->validated();

$booking = $this->bookingService->createBooking($data);

return response()->json($booking, 201);

}

}

// app/Repositories/BookingRepository.php

namespace App\Repositories;

use App\Models\Booking;

class BookingRepository

{

public function create(array $data): Booking

{

return Booking::create([

'user\_id' => $data['user\_id'],

'vehicle\_id' => $data['vehicle\_id'],

'start\_time' => $data['start\_time'],

'end\_time' => $data['end\_time'],

]);

}

public function attachServices(Booking $booking, array $serviceIds): void

{

$booking->services()->sync($serviceIds);

}

}

Ebben a mintában a BookingController csupán adatot kérdez le és továbbítja a BookingService-nek. A BookingService kezeli a tranzakciót, az üzleti szabályokat és a hívásokat a BookingRepository-re, amely a tényleges Eloquent-műveleteket végzi.

### 12.1.2 FormRequest validáció

A kérések validálását külön FormRequest osztályokkal oldjuk meg. Az alábbi példa a foglalás létrehozásához szükséges szabályokat tartalmazza.

**// app/Http/Requests/StoreBookingRequest.php**

**namespace App\Http\Requests;**

**use Illuminate\Foundation\Http\FormRequest;**

**use Illuminate\Validation\Rule;**

**class StoreBookingRequest extends FormRequest**

**{**

**public function authorize(): bool**

**{**

**// Csak ha be van jelentkezve**

**return auth()->check();**

**}**

**public function rules(): array**

**{**

**return [**

**'user\_id' => ['required', 'exists:users,id'],**

**'vehicle\_id' => ['required', 'exists:vehicles,id'],**

**'service\_ids' => ['required', 'array', 'min:1'],**

**'service\_ids.\*'=> ['integer', 'exists:services,id'],**

**'start\_time' => ['required', 'date', 'after:now'],**

**'end\_time' => ['required', 'date', 'after:start\_time'],**

**];**

**}**

**public function messages(): array**

**{**

**return [**

**'start\_time.after' => 'A kezdő időpontnak a jövőben kell lennie.',**

**'service\_ids.min' => 'Legalább egy szolgáltatást ki kell választani.',**

**];**

**}**

**}**

A controllerben egyszerűen típushinteljük a requestet, és a validált adatok a $request->validated() tömbben érhetők el.

## 1. Projektindító adatlap

Ez az oldal szolgál a projekt hivatalos induló pontjaként. Itt rögzítjük az alapadatokat, a felelősöket, a hatókört és a mérföldköveket. A projektindító adatlap kitöltése biztosítja, hogy minden érintett ugyanarról a kiindulóponti állapotról induljon, és később vissza lehessen térni a kezdeti dokumentumhoz.

A dokumentum bevezető részében röviden összefoglaljuk a projekt célját és üzleti kontextusát. A második blokkban a felelős szerepköröket nevezzük meg, és az ő feladatukat is megadjuk. A harmadik szekcióban a hatókör (scope) pontjait soroljuk fel, ügyelve arra, hogy mi tartozik bele, és mi nem. Végül az időtervet és a kapcsolódó mérföldköveket ütemezzük be konkrét dátumokkal.

| **Mező neve** | **Részletes leírás** | **Kitöltési útmutató** |
| --- | --- | --- |
| Projekt neve | A rendszerről, modulról vagy alprojektről használt hivatalos elnevezés | Írjuk pontosan, ahogy a megrendelővel egyeztettük |
| Verzió / Revízió | A dokumentum verziószáma (pl. 1.0, 1.1 …) | Minden szerkesztés után növeljük |
| Dokumentum dátuma | A kitöltés napja | év-hónap-nap formátumban |
| Projektgazda | A megrendelő / tulajdonos neve | Teljes név, beosztás |
| Projektmenedzser | A belső vezető, aki a projekt előrehaladásáért felel | Teljes név és elérhetőség (email/telefon) |
| Csapattagok | A fejlesztői, tesztelői és DevOps-tagok felsorolása | Név (Szerepkör) – pl. “Kovács Anna (Backend fejlesztő)” |
| Projektcélok | Rövid, mérhető eredmények felsorolása (SMART módon) | Pl. “Elérhető online foglalás 99,5%-os rendelkezésre állással” |
| Hatókör (Scope) | Mit tartalmaz a projekt (funkciók, komponensek); mi benne nem szerepel | Pontos lista, szükség esetén mellékletként bővítve |
| Mérföldkövek | Fontos átadások időpontjai (terv szerint) | Átadás neve – dátum – felelős |
| Kockázatok áttekintése | A kezdeti kockázatok összefoglalása (részletes űrlapra hivatkozva) | Hivatkozás a részletes kockázatkezelési lapra |

## 2. Persona kitöltő lap

A persona sablon lehetővé teszi, hogy a rendszer tervezése során háromszinten is megértsük a felhasználót: kik ők, mi motiválja őket, milyen a napi rutinjuk és mi okozza számukra a legnagyobb fájdalmat. Egy jól kitöltött persona segít a funkciók rangsorolásában, a felület tervezésében és a kommunikáció hangnemének kialakításában.

Mielőtt nekilátnánk a mezők kitöltésének, szánjunk időt interjúk vagy kérdőívek elemzésére. Szerezzünk be tényleges ügyféladatokat, nézzük át a korábbi support jegyeket, hogy a persona ne pusztán feltételezés vagy sztereotípia legyen, hanem valós adatokra épüljön.

| **Mező neve** | **Részletes leírás** | **Kitöltési útmutató** |
| --- | --- | --- |
| Persona neve | Fiktív, könnyen megjegyezhető elnevezés | Pl. “Sára, a rohanó dolgozó” |
| Kép és demográfia | Képviselt korosztály, nem, lakhely, munkahely | életkor 30–45, Budapest, irodai menedzser |
| Célok / motivációk | Mit szeretne elérni; mi az üzleti vagy személyes előny számára | Gyors, 0–24 órás foglalás, emlékeztetők, adatok újrafelhasználása |
| Fájdalompontok | Milyen problémákkal találkozik a jelenlegi folyamatban | Pontatlan foglalás, több hívás, várakozási idők |
| Technikai affinitás | Mennyire jártas a web- és mobilalkalmazások használatában (alapszint, haladó, profi) | Közepes – napi szinten használ applikációkat, de nem programozó |
| Napi rutin | Rövid leírás arról, hogyan telik egy munkanapja | 8–17: projektmegbeszélések, 17–19: hazavezetés közben mobilon intézi az ügyeket |
| Kedvenc kommunikáció | Milyen csatornát részesít előnyben (SMS, e-mail, push) | SMS – mert azonnal elolvassa |

## 3. Use Case űrlap

A use case sablon biztosítja, hogy minden funkció egyértelműen definiált legyen – ki kezdeményezi, milyen előfeltételeknek kell teljesülni, mi a fő eseményfolyam, és milyen alternatív útvonalak léteznek. A részletes űrlap megakadályozza, hogy a követelmények implicit, félreérthető vagy hiányos formában maradjanak.

Minden egyes use case kitöltése előtt mérjük fel a kapcsolódó üzleti szabályokat és a többi use case-el való átfedést. Az alt utakat nem csak hibakezelésre használjuk, hanem például arra is, ha a felhasználó közben kilép vagy módosít.

| **Mező neve** | **Részletes leírás** | **Kitöltési útmutató** |
| --- | --- | --- |
| Use Case azonosító | Egyedi kód vagy név | Pl. UC-03 Időpontfoglalás |
| Leírás (cél) | Rövid, egy mondatos összefoglaló | “Az ügyfél időpontot foglal szolgáltatásra.” |
| Szereplők (actors) | Ki kezdeményezi (ügyfél, recepciós, rendszer) | Entityvezetéssel – pl. “Felhasználó (actor), Backend (system)” |
| Előfeltételek (pre) | Mi kell ahhoz, hogy elinduljon (hitelesítés, jármű regisztrálva) | “Felhasználó bejelentkezett, legalább egy jármű regisztrált.” |
| Trigger | Mi az esemény, ami aktiválja (pl. gombnyomás) | “Foglalás gombra kattintás a dashboardon.” |
| Fő eseményfolyam | Lépésről lépésre, számozottan | 1. Szolgáltatás-választás<br>2. Jármű kiválasztás… |
| Alternatív utak | Minden elágazás, hiba- és módosítási eset részletezése | 3a. Nincs elérhető időpont – ajánlás lista <br> 5a. Hibás adat – visszajelzés |
| Utófeltételek (post) | Mit érünk el a sikeres vagy sikertelen végén | “Foglalás rögzítve, értesítő elküldve; ha nem, hibaüzenet.” |
| Kapcsolódó diagramok | Use case, aktivitás, szekvencia hivatkozása (oldalszám vagy fájlnév) | “Lásd UML-UC03.png és Activity-UC03.drawio” |

## 4. Teszt terv űrlap

Ez az űrlap a projekt tesztelési stratégiáját, felelősöket, eszközöket és határidőket foglalja össze. Részletes kitöltésével pontosan nyomon követhető, hogy mely komponenseket, milyen módszerekkel vizsgálunk, és milyen elfogadási kritériumokat alkalmazunk.

A sablon kitöltése előtt gyűjtsük össze a funkciólistát és a nem funkcionális követelményeket, hogy minden teszteset igazodjon a dokumentált elvárásokhoz. A környezeteknél írjuk le, hol futnak a tesztek (dev, test, staging) és milyen adatbázis-seed-ek kellenek hozzá.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Tesztazonosító** | **Teszttípus** | **Célkitűzés** | **Lépések** | **Elvárt eredmény** | **Felelős** | **Határidő** | | --- | --- | --- | --- | --- | --- | --- | | T-01 | Unit teszt | BookingService.createBooking funkció ellenőrzése | 1. Mock repo<br>2. Teszt adatok<br>3. Hívás | Hibamentes tranzakció, adatbázis rekord | Backend QA | 2025-10-01 | | T-02 | Integrációs teszt | API endpoint GET /services | 1. Staging GET hívás<br>2. JSON séma | JSON valid, státusz 200 | Integrációs QA | 2025-10-03 | | T-03 | End-to-end (Cypress) | Foglalási folyamat | 1. Belépés<br>2. Szolgáltatás választás<br>3. Foglalás | UI visszaigazolás, e-mail küldés | E2E QA | 2025-10-05 | | T-04 | Load teszt | 500 párhuzamos booking kérések kezelése | JMeter script futtatása | <300 ms válaszidő, 0 Error | Performance Eng | 2025-10-07 | |

## 5. Deployment ellenőrző lista

A deploy checklist segít abban, hogy minden frissítés során ugyanazokat a lépéseket hajtsuk végre, minimálisra csökkentve az emberi hibákat. A lista kitöltésével dokumentáljuk, ki, mikor végezte el a lépést, és rögtön látszik, ha valami kimaradt.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Lépés** | **Leírás** | **Teljesítve (I/N)** | **Felelős** |  | | --- | --- | --- | --- | --- | | Kód letöltése | git pull origin main |  | DevOps Eng. |  | | Unit tesztek futtatása | vendor/bin/phpunit && npm test |  | CI pipeline |  | | Build és képek tolása | docker build ... && docker push ... |  | CI pipeline |  | | Staging deploy | helm upgrade --install booking . |  | DevOps Eng. |  | | Post-deploy smoke teszt | /api/health, frontend load |  | QA |  | | Canary forgalomirányítás | 10%–50%–100% átváltás |  | DevOps Eng. |  | | Éles migrációk lefuttatása | php artisan migrate --force |  | Backend Dev |  | | Monitoring ellenőrzés | Grafana, Prometheus alert-ok |  | Ops |  | | Rollback jegy létrehozása | Hibák esetén rollback tervezet és PR |  | DevOps Eng. |  | |

Az Autószerviz Időpontfoglaló Rendszer legfőbb célja, hogy a műhely munkafolyamatait teljesen digitalizálja és automatizálja, ezáltal radikálisan csökkentve a manuális adminisztrációs terheket. A projekt üzleti értékének gerincét a 0–24 órás elérhetőség, a valós idejű kapacitáskezelés és az automatikus értesítési rendszer alkotja, amelyek együttesen garantálják, hogy egyetlen potenciális ügyfél se veszítse el a kapcsolatot a szervizzel. Ez nem pusztán egy technológiai fejlesztés, hanem egy olyan stratégiaváltás, amely lehetővé teszi a versenyképesség fenntartását és növelését a digitalizáció által diktált új piaci szabályok között. A projekt célkitűzései közé tartozik a visszajelzési ciklusok felgyorsítása, a hibaarány minimálisra csökkentése és a felhasználói elégedettség folyamatos mérési és javítási mechanizmusának kialakítása.

Stratégiai szinten a rendszer a vállalat digitális transzformációjának kulcseleme: a meglévő ügyfélkör megtartása mellett új, magasabb technológiai elvárásokat támasztó piacrészek felé nyit. A bevezetés gazdasági és üzemeltetési előnyeit hosszú távon a csökkenő emberi erőforrás-költségek, az automatikus naptárszinkronizációból eredő időnyereség és a precíz készletgazdálkodás együttes hatása garantálja. A projekt illeszkedik a vállalat fenntarthatósági törekvéseihez is: papírfelhasználás csökkenése, üzemi adminisztrációs költségek mérséklődése, valamint a hybrid munkavégzést támogató mobil hozzáférés mind olyan szempontok, amelyek erősítik a vállalati image-t és környezettudatossági üzenetet is közvetítenek. A kitűzött KPI-k – mint a rendszeres havi foglalási számok, a hibás foglalások aránya és a válaszidők – folyamatosan mérhetők, így a projekt eredményei gyorsan visszamérhetők a vállalati célrendszer felé.

A rendszer háromrétegű architektúrája világosan elkülöníti a backend, a frontend és az adatperzisztencia rétegeit, biztosítva ezzel a modularitást és a könnyű karbantarthatóságot. A backend a Laravel keretrendszeren alapuló RESTful API, amely kezeli az üzleti logikát, a hitelesítést és az értesítési feladatokat. A frontend React alapú SPA, amely asztali és mobil böngészőkben egyaránt natív élményt nyújt. A MySQL adatbázis normalizált struktúrája garantálja az adatintegritást, míg a Redis cache használata gyorsítja a gyakori lekérdezéseket és csökkenti a válaszidőt. A konténerizáció Dockerrel és futtatás Kubernetes klaszterben lehetővé teszi a vízszintes skálázást, az automatikus újrapróbálkozásokat és a magas rendelkezésre állást.

Az architektúra tervezésekor különös hangsúlyt fektettünk az API-versioning-re, így a jövőbeli bővítések sem rontják el a korábbi kliensalkalmazások működését. A kommunikáció teljes egészében HTTPS alapú, a JWT-alapú autentikáció pedig szerepkör-alapú jogosultságkezelést lát el minden végponton. A mikroservice-architektúra helyett monolitikus, modulokra bontott szolgáltatásban gondolkodtunk, mert ez a haladási fázis helyes egyensúlyát jelenti a fejlesztési sebesség és a bonyolultság között. A Kubernetes deployment integrációs folyamatokba épített health check és readiness probe mechanizmusok mellett Blue–Green deploy stratégiával csökkti a frissítések alatti kockázatot, és lehetővé teszi a rollback-et egy gombnyomással is.

A fejlesztést Scrum módszertan szerint szervezzük hat sprintre bontott 12 hetes iterációkban, ahol minden sprint két hétig tart. A sprinttervező ülések során a Product Owner mutatja be a legmagasabb prioritású backlog-elemeket, a csapat pedig story pointokban becsüli a feladatokat. A napi standupok folyamatos átláthatóságot biztosítanak, és azonnal felszínre hozzák az esetleges akadályokat. Minden sprint végén sprintdemót tartunk, ahol a stakeholder-ek élő bemutatón láthatják az elkészült funkciókat, és visszajelzést adhatnak a következő iterációhoz. A retrospective ülések során a team retrospektív gyakorlatokkal azonosítja a fejlesztési folyamat erősségeit és gyengeségeit, így a tanulságokat folyamatosan beépítjük a következő sprintbe.

A kódminőségért a SonarQube statikus elemző rendszer és a PHP-CS-Fixer, ESLint eszközök felelnek, amelyeket minden pull request előtt automatikusan lefuttatunk. A peer code review kötelező minden merge előtt, így a kód olvashatósága, tesztelhetősége és a SOLID-elvek betartása garantált. A CI pipeline GitHub Actions alapú, ahol a lintelés, unit és integrációs tesztek, valamint a Docker-image build és küldés a konténer-regisztrációs szolgáltatóba sorban lefutnak. A staging deploy mindezt tesztkörnyezetben elérhetővé teszi, ahol e2e tesztek is felülvizsgálják a rendszert, mielőtt a terméket élesítésre ajánlanánk.

A minőségbiztosítás több szintű tesztelési stratégiára épül: unit tesztek, integrációs tesztek és end-to-end automatizált forgatókönyvek váltják egymást a CI folyamatban. A unit tesztek PHPUnit és Jest alapon ellenőrzik az egyedi komponensek üzleti logikáját és funkcióit izolált környezetben, mocking technikákkal kiszűrve a külső függőségeket. Az integrációs tesztek élő SQLite- vagy staging adatbázissal futnak, ellenőrizve az API-végpontok, az adatbázis-repozitórium és a hitelesítési middleware együttműködését. Az e2e Cypress forgatókönyvek valós felhasználói utakat szimulálnak a böngészőben: regisztráció, belépés, foglalás, módosítás és törlés, így a teljes felhasználói élményt tesztelik, beleértve az automatikus értesítők küldését és a naptárszinkronizációt is.

A tesztelés során gyűjtött metrikák – kódlefedettség, hiba- és bugarány, tesztfuttatási idő – folyamatosan monitorozottak a SonarQube dashboardon és a CI pipeline riportjaiban. A kritikus vagy ismétlődő hibák automatikusan backlogba kerülnek, és prioritással kerülnek visszafejlesztésre. A tesztkörnyezetek minden sprint elején frissülnek az éppen stabilizált kódbázissal, így a QA csapat mindig a legfrissebb verzióval dolgozik. A teljes tesztelési dokumentáció tartalmazza a forgatókönyvek részletes leírását, a szempontokat és az elfogadási kritériumokat, hogy a vizsgabizottság vagy a minőségügyi szakértők bármikor visszakereshessék a teszteseteket és eredményeiket.

A telepítési folyamat a CI/CD pipeline végleges lépése, amely a main ágra való push és a manuális jóváhagyás után automatikusan elindul. A GitHub Actions-ban definiált workflow először lefuttatja a teljes tesztcsomagot, majd elkészíti és tolja a Docker-image-eket a privát registry-be. A staging környezetbe való telepítést Helm chartok segítségével végezzük, amelyek a szükséges ConfigMap és Secret erőforrásokat is konfigurálják. A readiness és liveness probe-ok biztosítják, hogy a konténerek csak akkor vegyék át a forgalmat, ha valóban egészségesek, így elkerülhetőek a félkész állapotok miatti szolgáltatáskiesések.

Éles környezetben Blue–Green deploy vagy Canary release stratégia alkalmazásával vezetjük be az új verziókat, lehetővé téve a rollback-et egyetlen parancs segítségével. A környezetek – dev, test, staging, prod – elkülönített namespace-ekben futnak, és dedikált erőforrás-kváta biztosítja, hogy a tesztkörnyezetek ne versenyezzenek a termeléssel. A monitoring és alerting rendszerek – Prometheus, Grafana, ELK – folyamatosan gyűjtik a metrikákat és naplókat, riasztásokat generálva a felelős üzemeltetők számára. A backup és restore rutinok automatizáltan futnak, napi teljes adatbázismentést és óránkénti tranzakciós napló-mentést biztosítva, 24 órás RTO mellett.

A bevezetett rendszer keretében a következő fejlesztési irányok a legnagyobb üzleti értéket ígérik. Első körben egy üzleti intelligencia modul integrálását tervezzük, amely vizualizálja a foglalási trendeket, a legnépszerűbb szolgáltatásokat és a csúcsidőpontokat, így a menedzsment adatvezérelt döntéseket hozhat. Ezt egy chatbot-integráció követi, amely a tipikus ügyféligények 70–80 százalékát automatizálva kezeli – foglalás, módosítás, törlés –, miközben a beszédalapú, természetes nyelvi feldolgozás lehetővé teszi a zökkenőmentes felhasználói élményt.

# 13. Záróoldal

Ez a dokumentáció részletesen feltárta az Autószerviz Időpontfoglaló Rendszer teljes spektrumát, kezdve a piaci háttér és üzleti kontextus feltérképezésétől, a funkcionális és nem funkcionális követelmények aprólékos definiálásán át a rendszerterv, fejlesztés, tesztelés és üzemeltetés minden lépéséig. A bevezetéstől a finomhangolásig mindent lefedtünk: megmutattuk, hogyan épül fel egy valódi, életszerű problémára reagáló webes és mobil platform, miként valósulnak meg a RESTful API hívások, hogyan szolgálja ki a rendszer a felhasználókat 0–24 órában, és miként gondoskodtunk a karbantarthatóságról, a skálázhatóságról, valamint a folyamatos minőségbiztosításról.

A projekt során végig a tiszta kód elveit követtük: a Controller–Service–Repository rétegek elkülönítésével, a FormRequest osztályokkal definiált validációs szabályokkal és az automatikus Queue Job-ok menedzselésével biztosítottuk, hogy minden funkció átlátható, tesztelhető és könnyen bővíthető legyen. A frontendoldalon a React komponensek moduláris felépítésével és a styled-components által vezérelt témarendszerrel értük el, hogy asztali és mobil eszközökön egyaránt natív jellegű felhasználói élmény jöjjön létre.

A CI/CD pipeline-unknak köszönhetően minden egyes commit után automatikus lintelés, tesztfuttatás, Docker-kép build és staging környezetbe való telepítés zajlik le emberi beavatkozás nélkül. Ez a folytonos integráció és deploy nem csak hibamentességet és gyors visszajelzést garantál, hanem lehetővé teszi, hogy a legkisebb módosítások is azonnal éles próba alá kerüljenek. Mindezek eredőjeként olyan fejlett infrastruktúrát alakítottunk ki, amely Kubernetes klaszterben futtatva képes dinamikusan méretezni a kiszolgáló egységeket, és 99,9 % feletti rendelkezésre állást biztosítani.

A rendszer legfontosabb eredménye a felhasználói élmény radikális javulása: az ügyfelek percek alatt, különösebb ügyintézés nélkül foglalhatnak időpontot a szervizbe, miközben a naptárszinkronizáció és az automatikus SMS- és e-mail emlékeztetők gondoskodnak arról, hogy senki ne feledkezzen el a beosztott időpontról. Ez a tökéletes összehangoltság a foglalási folyamat és az üzemeltetői adminisztráció között a korábbi papír- és telefonközpontú megoldásokkal ellentétben a hibák számát közel nullára csökkentette, miközben növelte a műhely kapacitáskihasználását és ügyfélhűségét.

Az adattárolás és adatkezelés teljesen GDPR-kompatibilis keretek között zajlik. A személyes adatok titkosítása AES-256-tal, a kommunikáció TLS-en, a jogosultságkezelés pedig role-based access control segítségével történik. Így nemcsak technikailag védtük meg az érzékeny információkat, hanem megfelelünk a legszigorúbb európai adatvédelmi előírásoknak is. A naplózás, monitoring és auditálás teljes körűen dokumentált, a logok hat hónapig megőrződnek, és bármikor visszakövethető a rendszerben végrehajtott összes művelet.

A szerviz informatikai környezetének modernizálása túlmutat egy egyszerű foglalórendszeren: a konténerizált szolgáltatások, a Helm-chartok és Kubernetes-deploymentek révén a csapat azonnal átültethető más, hasonló volumenű ügyfélprojektekbe is. Az előre definált, verziózott infrastruktúra-sablonok csökkentik a bevezetési időt és a konfigurációs hibák kockázatát, így a jövőben bármilyen új telephelyen vagy partnernél néhány óra alatt működőképes környezet állítható fel.

A folyamatos karbantartás és üzemeltetés során bevezetett blue-green deploy, rollback szkriptek és automatizált backup stratégiák lehetővé teszik, hogy bárminemű frissítést biztonságosan, minimális leállással hajtsunk végre. A kódlefedettségi mutatók, a SonarQube-jelentések és a Cypress-tesztek eredményei mind egy központi dashboardon jelennek meg, így a műszaki csapat valós időben láthatja a rendszer egészségi állapotát és a biztonsági kockázatokat.

Az előttünk álló következő fejlesztési fázisokban a beépített üzleti intelligencia elemzőmotor segítségével részletes foglalási statisztikákat és trendelőrejelzéseket készítünk. Ez nem csupán azt mutatja meg, mely szolgáltatások a legnépszerűbbek, hanem azt is, mikor van csúcsidő, hogyan alakul a szerelői kihasználtság, és milyen időpontokra érdemes akciókat tervezni. A bevezetendő chatbot-modul a felhasználói kérdések 80 %-át automatizáltan kezeli majd, csökkentve az operátori terhelést és gyorsabb választ adva a tipikus foglalási, módosítási és törlési kérésekre.

A natív mobilalkalmazásba épített push notification szolgáltatás is hamarosan elérhetővé válik, amely nem csak a foglalás előtt küld emlékeztetőt, hanem a szervizben végzett műveletekről, kedvezményes ajánlatokról és szezonális karbantartási tippekről is azonnali, személyre szabott tájékoztatást ad. Ezzel a megoldással a felhasználói elköteleződést és az ismételt látogatások számát egyaránt növelni tudjuk.

Végül a CRM- és ERP-rendszerek integrációja lehetővé teszi, hogy az értékesítés, készletgazdálkodás és pénzügyi könyvelés közvetlenül a foglalásokból származó adatokra épüljön. Az IoT-eszközök – például okos diagnosztikai aljzatok és műhelyi szenzorok – bekötése pedig valós idejű adatokat szolgáltat a járművek állapotáról és a műhely kihasználtságáról, így a karbantartási ciklusok automatizáltan, prediktív alapon indíthatók el.

Szeretnénk köszönetet mondani mindenki­nek, aki részt vett e projekt sikerében: a megrendelői oldalon dolgozó szakértőknek, a projektmenedzsmentnek, a backend és frontend fejlesztőknek, a DevOps-mérnököknek, a QA- és tesztelési csapatnak, valamint a felhasználói visszajelzéseket gyűjtő operatív támogatóknak. Külön köszönjük az akadémiai mentoroknak és üzleti tanácsadóknak a konstruk­tív javaslatait, amelyek finomították a követelményeket és a megvalósítás részleteit.

Ezzel a záróoldallal fejezzük be a dokumentációt, de a közös munka csak most veszi kezdetét. Bízunk benne, hogy az Autószerviz Időpontfoglaló Rendszer hamarosan élőben bizonyítja megbízhatóságát, üzleti értékét és felhasználói élményét. A következő verziók újabb innovációt hoznak majd, és a rendszer folyamatos fejlődése révén az Önök műhelye mindig egy lépéssel a versenytársak előtt járhat.

Köszönjük, hogy végigolvasták ezt a dokumentumot, és örömmel várjuk a bevezetés utáni visszajelzéseket, kérdéseket és javaslatokat.

**Az Autószerviz Időpontfoglaló Rendszer csapata**