# Szemantikus verziókezelés (Semantic Versioning)

## Bevezetés

A modern informatikai rendszerekben – különösen mikroszolgáltatásos architektúrák, API-k és könyvtárak esetén – kiemelten fontos, hogy a különböző komponensek egymással kompatibilisek maradjanak. A szemantikus verziókezelés (Semantic Versioning, röviden SemVer) egy olyan szabványos megközelítés, amely a verziószámokon keresztül információt hordoz a változások természetéről.

A SemVer célja:

- a függőségek kezelhetőségének javítása
- a kompatibilitási problémák csökkentése
- az integráció megkönnyítése

## Alapelvek

A szemantikus verzió formátuma:

```
MAJOR.MINOR.PATCH
```

ahol:

| Elem | Jelentés | Mikor növeljük? |
|---|---|---|
| MAJOR | főverzió | inkompatibilis változás esetén |
| MINOR | alverzió | új, kompatibilis funkció esetén |
| PATCH | javítás | hibajavítás esetén |

A verziószám minden eleme nemnegatív egész szám, és növekvő sorrendben változik.

## A verziószám mögötti logika

A SemVer egyik legfontosabb gondolata, hogy a verziószám kommunikációs eszköz a fejlesztők között.

Példa:

1.2.3 → stabil API

1.2.4 → ugyanaz az API, csak hibajavítás

1.3.0 → új funkciók, de régi kód továbbra is működik

2.0.0 → fő verzió váltás → régi kód eltörhet

# Verziónövelés döntési folyamata

A verzió növelése nem véletlenszerű, hanem szabályalapú döntés.

flowchart TD A["Változás történt"] --> B{"Kompatibilis?"} B -- "Nem" --> C["MAJOR növelése"] B -- "Igen" --> D{"Új funkció?"} D -- "Igen" --> E["MINOR növelése"] D -- "Nem" --> F["PATCH növelése"]

# A három verziószint részletesen

## MAJOR verzió

A MAJOR verzió növelése azt jelenti, hogy a rendszerben olyan változás történt, amely nem kompatibilis visszafelé.

Példák:

- függvény paraméterlistájának megváltoztatása
- API endpoint törlése
- adatstruktúra módosítása

Ez a legkritikusabb változás, mert:

- a kliensek frissítés nélkül hibásan működhetnek
- integrációs hibák jelenhetnek meg

## MINOR verzió

A MINOR verzió növelése új funkciók bevezetését jelenti, amelyek nem törik el a meglévő működést.

Példák:

- új API endpoint hozzáadása
- opcionális paraméter bevezetése
- új szolgáltatás modul

Fontos:

- a régi kliensek továbbra is működnek
- a PATCH érték ilyenkor nullázódik

## PATCH verzió

A PATCH verzió növelése kizárólag hibajavításokra szolgál.

Példák:

- bug fix
- teljesítmény javítás
- dokumentáció pontosítása (ha nem érinti az API-t)

Ez a legbiztonságosabb frissítés típus.

# Verziók életciklusa

## 0.x.x – fejlesztési fázis

- nincs stabil API
- bármilyen változás történhet
- nem ajánlott éles rendszerben használni

## 1.0.0 – stabil kiadás

Ez az a pont, ahol:

- az API stabilnak tekinthető
- a SemVer szabályokat kötelező betartani

# Előkiadások és metaadatok

## Előkiadás (pre-release)

```
1.0.0-alpha
1.0.0-beta.1
1.0.0-rc.1
```

Jelentés:

- még nem stabil verzió
- tesztelésre szolgál
- alacsonyabb prioritású, mint a végleges verzió

## Build metaadat

```
1.0.0+build.123
```

Jellemzők:

- csak információ (pl. build szám)
- nem befolyásolja a verziók sorrendjét

# Verziók összehasonlítása

A verziók sorrendje a következő logika szerint történik:

MAJOR → MINOR → PATCH

majd pre-release

Fontos:

- a build metaadat nem számít
- az előkiadás mindig “gyengébb”, mint a normál verzió

# Fontos szabályok

- Egy kiadott verzió nem módosítható
- Minden változtatás új verziót igényel
- Kötelező a publikus API definiálása
- A verziószámok növekedése monoton

# Gyakorlati jelentőség az integrációban

A szemantikus verziókezelés különösen fontos az alábbi esetekben:

mikroszolgáltatások közötti kommunikáció

REST / GraphQL API-k használata

külső könyvtárak integrációja

CI/CD pipeline-ok

Előnyök:

automatikus dependency frissítés biztonságosan

kompatibilitási hibák csökkentése

jobb verziókövethetőség

# Példa verziók alakulására

```
 1.2.3 → bugfix → 1.2.4 1.2.3 → új feature → 1.3.0 1.2.3 → breaking change → 2.0.0
```

# Összefoglalás

A szemantikus verziókezelés egy egyszerű, de rendkívül hatékony módszer arra, hogy a szoftverek fejlődését strukturáltan és érthetően kövessük, miközben minimalizáljuk az integrációs kockázatokat és növeljük a rendszerek megbízhatóságát.