

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Készítette

Balog Ábel PéterKonzulens

Albert István

2016

Tartalomjegyzék

[Összefoglaló 5](#_Toc449771634)

[Abstract 6](#_Toc449771635)

[1. Bevezetés 7](#_Toc449771636)

[1.1. Gantt diagram 7](#_Toc449771637)

[1.2. Létező megoldások 7](#_Toc449771638)

[2. Vékonykliens fejlesztés 9](#_Toc449771639)

[2.1. Bevezetés 9](#_Toc449771640)

[2.2. JavaScript 9](#_Toc449771641)

[2.3. TypeScript 9](#_Toc449771642)

[2.4. HTML5, Canvas, KonvaJS 13](#_Toc449771643)

[2.5. AngularJS 13](#_Toc449771644)

[2.5.1. Angular Material 13](#_Toc449771645)

[3. Gantt diagram 15](#_Toc449771646)

[3.1. Bevezetés, követelmények 15](#_Toc449771647)

[3.2. Modell 15](#_Toc449771648)

[3.2.1. Projekt felépítése 16](#_Toc449771649)

[3.2.2. Munkanaptár 17](#_Toc449771650)

[3.2.3. Ütemező 17](#_Toc449771651)

[3.2.4. Erőforrások 17](#_Toc449771652)

[3.3. Ütemezés, CPM 17](#_Toc449771653)

[3.3.1. Bevezetés 17](#_Toc449771654)

[3.3.2. Kritikus útvonal módszer 17](#_Toc449771655)

[3.3.3. Erőforrások ütemezése 18](#_Toc449771656)

[3.4. Megvalósítás 19](#_Toc449771657)

[4. A teljes webalkalmazás 20](#_Toc449771658)

[4.1. Bevezetés, követelmények 20](#_Toc449771659)

[4.2. Architektúra 20](#_Toc449771660)

[4.3. Megvalósítás 20](#_Toc449771661)

[5. Összefoglalás 21](#_Toc449771662)

[Ábrák jegyzéke 22](#_Toc449771663)

[Táblázatok jegyzéke 23](#_Toc449771664)

[Irodalomjegyzék 24](#_Toc449771665)

[Függelék 25](#_Toc449771666)

Hallgatói nyilatkozat

Alulírott Balog Ábel Péter, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 04. 30.

Balog Ábel Péter

# Összefoglaló

A

# Abstract

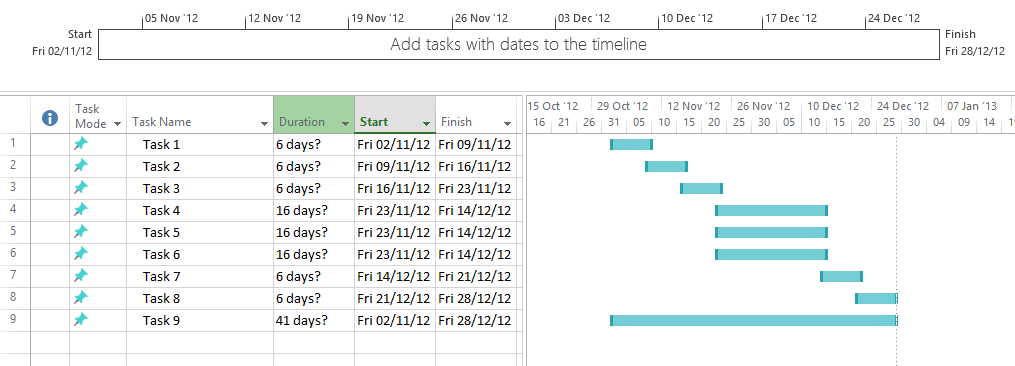
English abstract of the thesis work. This summarises the content of the thesis in 0.5–1 pages and is uploaded to the Thesis Work Portal as well.

# Bevezetés

## Gantt diagram

A Gantt diagram a projekt menedzsment egyik alapvető eszköze, ami vizuálisan, könnyen érthető módon és átláthatóan megjeleníti az egyes feladatok a projekt során, azok időszükségleteit, illetve hogy mikorra is ütemezzük annak elvégzését.

Megjelenésre egy oszlop diagramhoz hasonlítható, aminek egyes oszlopai jelölik, hogy az adott feladatot mikor kell elvégezni az ütemezés szerint.



1. ábra Gantt diagram a Microsoft Project 2013 programban

Karol Adamiecki volt az első, aki ilyen diagramot készített 1896-ban, de eredményeit nem publikálta. Később Henry Gantt volt az, aki tőle függetlenül kitalálta a ma ismert Gantt diagramot, amit róla neveztek el. Később Adamiecki is publikálta eredményeit, de azok csak Lengyelországban váltak ismertté.

Az informatika előretörésével kapott újra nagy hangsúlyt a diagram, hiszen a számítógépek rendkívül gyorsan tudták a feladatokat ütemezni, és azokat vizuálisan megjeleníteni.

Ma egy projektmenedzser munkájának jelentős része a projekt feladatainak ütemezésével, erőforrások feladathoz rendelésével és a feladatok közötti függőségek felismerésével telik, amikhez a Gantt diagram és a köré épült megoldások adják az eszközöket.

## Létező megoldások

Az egyik legelterjedtebb létező megoldás a Microsoft Project, így a dolgozat folyamán ezt tekintem mintának. Az 1. ábrán látható, hogy hogyan épül fel a felülete. Az egyes feladatok egymás alatt jelennek meg további információk társaságában (időtartam, kezdés, befejezés időpontja). Bal oldalt látható vizuálisan, hogy mely feladatokat mikor kell elvégezni. Látható, hogy az egyes feladatok csoportba rendezhetőek, és köztük függőségeket tudunk definiálni, amik vizuálisan is megjelennek.

Fontos szempont, hogy a diagram jól átlátható, az egyes feladatok jól elkülöníthetőek egymástól, és egy külső szemlélő számára is könnyen átlátható, hogy az adott projekt milyen részfeladatokból áll, azok között milyen tartalmazási és függőségi relációk vannak, illetve mikor kell őket elvégezni.

A diagramnak nem célja, hogy leírja az egyes feladatok részleteit, technológiai követelményeit, ezek az információk mind az átláthatóság kárára tudnának csak megjelenni, emiatt ezekkel kapcsolatban semmilyen információt nem hordoz.

# Vékonykliens fejlesztés

## Bevezetés

Az alkalmazás vékonykliens környezetben kell, hogy fusson, HTML5 és JavaScript technológiák segítségével. Hátránya, hogy a legtöbb esetben lassabb, mint a natív megoldások, viszont cserébe platform független, azaz minden környezetben futni fog, mindössze egy böngészőre van szükség. Mivel a feladat kiírásában szerepel, hogy ez egy önállóan futni képes alkalmazás legyen, ezért az üzleti logika teljes része kliens oldalon kell, hogy fusson.

## JavaScript

A fejlesztéshez a JavaScript nyelvet választottam. A támogatottsága nagyon széles a böngészők körében, illetve futtatásához nem szükséges külön beépülő modul.

A JavaScript-et 1997-ben ECMAScript néven szabványosították, itt írták le először egységesen a nyelv alapjait. Alapvetően egy objektum alapú nyelvről van szó, azaz tartalmaz objektumokat, de procedurálisan is lehet programozni. A JavaScript objektum felfogása erősen eltér a szokásos nyelvekétől (C++, Java, C#), úgynevezett prototípus alapú nyelvről beszélünk. Minden objektum egy prototípus példánya, de azt futási időben tetszőleges módosíthatjuk, adhatunk hozzá metódusokat, tagváltozókat, módosíthatjuk értéküket, típusukat, implementációjukat.

A nyelvben van beépített garbage collector, így nem kell különösen nagy figyelmet fordítani a memóriaszívárgásra, mindezek ellenére segíthetjük a futtató környezetet, például egy váltózó kinullozásával jelezhetjük, hogy már nem tartunk igényt az értékére.

## TypeScript

Igaz, hogy a JavaScript nyelv prototípus alapú objektum megközelítése nagyon jól támogatja a bővíthetőséget, de fejlesztési időben az egyes változók típusaival kapcsolatban semmilyen megkötés sincs, így a függvények, objektumok közötti kommunikációban, fordítási időben nem tudjuk megkövetelni a hívó féltől, hogy egy adott objektum rendelkezzen bizonyos tulajdonságokkal.

Ennek orvoslására több megoldás is született, amik különböző módokon segítik a fejlesztést. Az egyik ilyen a Microsoft által fejlesztett és karbantartott TypeScript. A fejlesztésnél fontos szempont volt, hogy lehetőleg hasonló szintaktikája legyen, mint a készülő EcmaScript 6-nak, ezzel segítve az esetleges későbbi átállást. A TypeScript nem egy új nyelv, csak egy plusz réteg a JavaScript felé, ami fejlesztési időben garantálja a típusosságot, JavaScript-re fordul, és futási időben már semmilyen szerepet sem játszik.

Egy egyszerű osztály így néz ki:

**module** Test {  
 **export class** Person {  
 **public name**: **string**;  
 **private parents**: Person[];  
   
 **constructor**() {  
 **this**.**name** = **""**;  
 **this**.**parents** = [];  
 }  
   
 addParent(person: Person): **void** {  
 **this**.**parents**.push(person);  
 }  
 }  
}

A *module* kulcsszóval tudjuk az osztályokat csoportokba rendezni, hasonló, mint a *Java*-ban a *package* vagy *C#*-ban a *namespace*. A modulon belül van lehetőség osztály deklarációra. Az *export* kulcsszó jelenti, hogy az osztály a modulon kívülről is elérhető. Az utána lévő kód nagyon hasonlít a szokásos programozási nyelvekben megismert szintaktikához, egyedül a típusdeklaráció más, kivételesen nem a változó neve elé írjuk a típust, hanem a változó után, kettősponttal elválasztva.

A fordító a TypeScript fájlokat lefordítja JavaScript nyelvre (esetleg létrehoz hozzá egy map fájlt, ami a debugger-nek segít, hogy melyik JavaScript kódrészlet melyik TypeScript kódrészlethez tartozik). Az így keletkezett fájlt már be tudjuk tenni a HTML-be, és a böngésző is képes a futtatására.

A TypeScript legnagyobb előnye, hogy kompatibilis a már létező JavaScript-ben írt könyvtárakkal. Ha valaminek az any típust adjuk, akkor az bármi lehet (azaz a fordító nem ellenőrzi, hogy a rajta meghívott metódusok, tagváltozók valóban részései az objektumnak), vagy pedig elég megadni a függvénykönyvtár interfészét, ami fejlesztési időben segít a kódkiegészítéssel, futási időben pedig már az interfész egyes kulcsszavai pontosan egy-egy JavaScript függvényt/típust takar. A nagyobb függvénykönyvtárak vagy saját maguk készítenek ilyen interfészleírást, vagy pedig a fejlesztői közösség készíti el őket, és teszik elérhetővé bárki számára.

A nagy előnye egy ilyen plusz réteg bevezetésének, hogy fejlesztés közben sokkal kevesebbszer kell belenézni a dokumentációba vagy a hívott kód implementációjába, hiszen a fejlesztési környezet az automatikus kiegészítéssel segíti a fejlesztést. Ha az egész alkalmazás TypeScript-ben íródott, akkor biztosak lehetünk abban, hogy ahol egy adott típust megkövetelünk, ott az is van, de ha már külső JavaScript kódból hívjuk, akkor szükséges az ellenőrzés.

## HTML5, Canvas, KonvaJS

## AngularJS

A webalkalmazás működését a Google által fejlesztett AngularJS fogja segíteni. Kezdetekben MVC (Model-View-Controller) mintát követett, idővel viszont már sok MVVM-es (Model-View-Viewmodel) minta is része lett a technológiának, így most már az általuk kitalált MVW (Model-View-Whatever) mintát követik.

Ez a keretrendszer a fejlesztőt igyekszik abban segíteni, hogy a kezdektől fogva támogassa

Az AngularJS legnagyobb előnye, hogy a HTML bizonyos elemeit hozzá lehet kötni (binding – adatkötés) a mögötte álló Controller bizonyos elemeihez, és amíg az AngularJS kereteiben belül dolgozunk, addig a változások automatikusan nyomon lesznek követve.

Az AngularJS funkcionalitása elég sokrétű, ezért nem is témája a jelen dolgozatnak, de egy jól bővíthető, dinamikusan fejlődő függvénykönyvtárról van szó, ami segít abban, hogy az alkalmazás minél nagyobb része (jelen esetben az egész) JavaScript-ben kerüljön implementálásra. Tartalmaz template könyvtárat, REST klienst és rengeteg egyéb funkciót.

### Angular Material

Az Angular Material szintén a Google által fejlesztett függvénykönyvtár, ami az egységes, reszponzív felület kialakításában nyújt segítséget a fejlesztő számára.

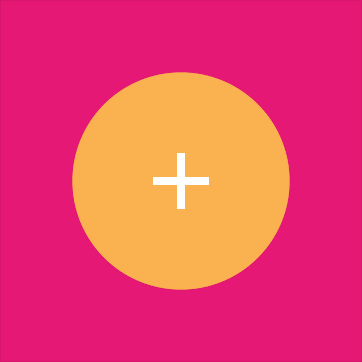
Az Android mobil operációs rendszerének Lollipop (5.0) kiadásában a Google egy új designelvet vezetett be, ezt hívják Material Design-nak. A következő alapelveket fekteti le:

1. Az anyag mindennek az alapja: A designelemek alapvetően papírból (rétegek) épülnek fel, amiknek vannak színei. Ezek a rétegek egymáson helyezkedhetnek el, egymás fölött és alatt, van szélességük, magasságuk, de mélységük nincs, viszont a távolságuk a felhasználótól különbözhetnek egymástól (árnyékként jelenik meg). Fontos, hogy a felület megjelenése közel álljon a valósághoz, azaz például az egyes réteget egymást nem metszhetik.



2. ábra Material design – anyag

1. Merészség, grafikusság, tudatosság: A felület színvilága legyen merész, megfontolt. Legyenek grafikus elemek, változó formák. Fontos, hogy a felületet tudatosan építsük fel, a felhasználónak a tekintetét a fontos helyekre „csábítsa”. A lényeg legyen kiemelve, ne kelljen keresgélni, a felület legyen szemcsalogató.



3. ábra Material Design – merészség

1. Az animáció, mozgás fontossága: A felhasználók reakciója egy mozgásra általában egy inflekciós pontja, azaz a felhasználói műveletek a mozgások irányát módosítják. A mozgások folyamatosak legyenek, semmi se törje meg a folytonosságot, az animáció kezdeti szakaszát gyorsulás, a közepét egységes sebesség a végét pedig lassulás jellemezze.

A fenti elvek betartásában nyújt segítséget az Angular Material. Különböző modulokat tartalmaz, amik kész komponensként beépíthetőek a webalkalmazásba. A modulok beszúrása az AngularJS által kitalált direktívák segítségével működik, azaz vagy teljesen saját fejlesztésű HTML tag-eket használatával, vagy egyedi attribútumok használatával lehet testre szabni a megjelenést, viselkedést.

Segítséget nyújt, hogy az alkalmazás reszponzív működésű legyen, azaz a felület a képernyő vagy felbontás függvényében változzon, így igazodva például a mobil kijelzőkhöz.

# Gantt diagram

## Bevezetés, követelmények

Ebben a részben azt fogom kifejteni, hogy a diagramnak és ütemezésnek milyen követelményeknek kell hogy megfeleljen, és ezeket hogy valósítottam meg.

Mint, ahogy az 1. fejezetben kifejtettem a cél a Microsoft Project 2013 alapján egy Gantt diagram megjelenítő, kezelő és ütemező elkészítése. Első körben az ütemezés és a megjelenítés megvalósítását fogom ismertetni.

## Modell

C:\Users\Abel\Downloads\overview.png

A modell négy darab komponensből épül fel, amik egymást teljes mértékben ismerik, és szorosan kapcsoltak.

### Projekt felépítése

C:\Users\Abel\Downloads\model (2).png

A projekt és az abban lévő feladatok kapcsolata a fenti ábrán látható

* Project: egy projektet modellező osztály. A legfontosabb

### Munkanaptár

### Ütemező

### Erőforrások

## Ütemezés, CPM

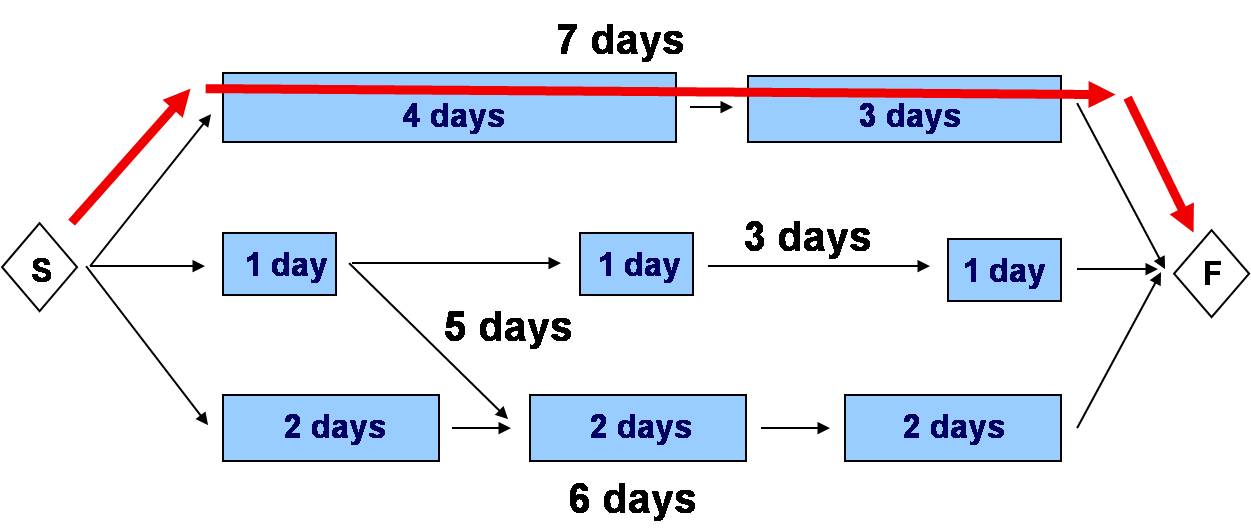
### Bevezetés

A Gantt diagram nem arról szól, hogy hogyan ütemezzük a feladatokat, hanem, hogy az eredményt hogyan jelenítsük meg a felhasználó számára. Ez az elképzelés hosszú ideig megállta a helyét, az ütemezés humán feladat volt, egészen addig, amíg a számítógépek számítási teljesítménye kellően megnövekedett ahhoz, hogy ezt automatikusan és rövid idő alatt elvégezzék.

A feladat megoldására jelenleg nem létezik olyan algoritmus, ami tökéletes eredményt adna és nem NP nehéz (<https://mialmanach.mit.bme.hu/aima/ch12s01>). A legelterjedtebb megold

### Kritikus útvonal módszer

Abban az esetben, ha előre ismerjük a feladatok végrehajtási idejét, azok erőforrás igényeit és a függőségeket, akkor első lépésként meg kell állapítanunk, hogy a feladatok lehetséges kezdési és befejezési idejét. Ezt a kritikus útvonal módszerrel tehetjük meg (CPM – Critical Path Method), amit az alábbi ábra szemléltet:



A teljes feladatsor kezdését az S (Start), míg a befejezését az F (Finish) jelzésű rombusz jelöli, közöttük kék téglalapokban láthatóak a feladatok, benne azok végrehajtási idöjével, a feladat közötti függőségeket pedig a nyilak mutatják. A nagy fekete feliratok jelölik az egyes utak (ha gráfként képzeljük el a diagramot) végrahajtási idejét.

A módszer első lépéseként megallapítjuk az előbb említett végrehajtási időket. Ha ezek megvannak, akkor kiválasztjuk a leghosszabb utat, amit kritikus útnak nevezünk (critical path, piros színnel jelölve az ábrán). Továbbiakban bármilyen ütemezőt is használjunk, a legfontosabb, hogy a kritikus úton lévő feladatokat elsőként hajtsuk végre. Könnyen belátható, hogy ez miért van így, hiszen a projekt szűk keresztmetszetét ezek a feladatok adják, ennél az útnál rövidebb idő alatt a projekt nem fejezhető be.

Amiről a CPM nem beszél, az az, hogy a többi feladatot milyen sorrendben hajtsuk végre. Erre a kérdésre a „legkevesebb tartalék először” (Least Slack Time First) módszer ad egy jó megoldást.

Az algoritmus alapját a következő heurisztika adja: végezzük el a rendelkezésre álló feladatok közül azt, amelyiknek a csúszása legnagyobb hatással van a projekt befejezési idejére, azaz melyik feladat halogatható a legkevésbé. A halogathatóság (slack time) kiszámításához minden feladathoz meg kell ismernünk, hogy mi az a legkorábbi és legkésőbbi kezdés illetve befejezés, ami még nem okoz csúszást a teljes projekt miatt. Az legkorábbi értékek a kritikus út kiszámítása közben maguktól adódnak, viszont a legkésőbbi értékek már nem adódnak ilyen egyszerűen.

Ahhoz, hogy ezek az értékek kijöjjenek, újra le kell futtatni a CPM algoritmust, most viszont visszafelé. Ez azt jelenti, hogy a projekt végén kezdjük a számítást, megfordítjuk a függőségek irányát, kezdési időpontnak beállítjuk az utolsó feladat legkorábbi befejezési időpontot (amit már ismerünk), és így indítjuk el az algoritmust.

Mikor már minden adatot ismerünk, akkor a slack time már kiszámolható a legkorábbi és legkésőbbi kezdési/befejezési időpont különbsége alapján.

### Erőforrások ütemezése

Az eddig leírt folyamat nem veszi figyelembe, hogy az erőforrások korlátosok, mert erről nem szól CPM és ahhoz kapcsolódó algoritmusok. Amit kaptunk, az az, hogy minden időpillanatban meg tudjuk mondani, hogy melyik feladat elvégzése a legjobb. Ahhoz, hogy ezt be tudjuk építeni az algoritmusba, el kell dönteni, hogy preemptív ütemezőt szeretnénk-e. Mivel, az hogy megszakítsuk egy feladat elvégzését, és egy másikkal folytassuk a sorozatot az jelent egy kis plusz időt, míg a tevékenység végzője átáll. Hogy ezt az időt beszámítsuk, nagyon megbonyolítaná az algoritmust, és ha kellően kis feladatokra bontjuk a projektet, akkor nem lenne gyorsabb egy preemptív ütemező.

Az általam választott megoldás az alábbi módon néz ki. Minden erőforráshoz annyi virtuális idővonal (továbbiakban foglaltság) kerül kialakításra, ahogy darab egység rendelkezésre áll. Ezek után kiválasztom az első olyan feladatot, aminek az összes függősége elkészült már, majd ráillesztem ezekre a foglaltságokra, mindegyik erőforrásból annyira amennyire szüksége van.



A fenti ábrán színessel vannak jelölve a már foglalt időszakok egy erőforráshoz. Ha a következő feladathoz két darab számítógépre (Computer) és egy darab fejlesztőre (Developer) van szükség, akkor őt már csak a kék időszak után tudjuk elvégezni. Ha viszont utána jön egy feladat, ami nem annyira fontos (a tartalék, vagy slack time nagyobb), de csak egy fejlesztőre van szükség az elvégzéséhez, akkor azt el is kezdhetjük rögtön, és annak ellenére, hogy nem fontos annyira, mégis előbbre kerül mint a fontosabb.

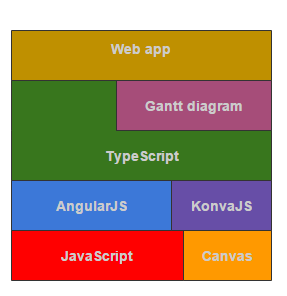
A fent leírt módszerrel már teljesen le van fedve az ütemezés problémája, mind az erőforrások mind a függőségek tekintetében.

## Megvalósítás

# A teljes webalkalmazás

## Bevezetés, követelmények

## Architektúra



4. ábra Architektúra

Fent látható az elkészült szoftver architektúrájának egy egyszerűsített ábrája. Az alsó rétegen a JavaScript és Canvas helyezkedik fel. Ezek olyan technológiák, amit a böngészők natív módon támogatnak, azaz semmilyen plusz könyvtár vagy beépülő modul használata nem szükséges. A KonvaJS a Canvas-t elfedő JavaScript függvénykönyvtár, míg az AngularJS magáétz az egész webalkalmazás működéséért és megjelenéséért felelős eszköz.

A Web app és a Gantt diagram az általam írt komponensek, amik mind TypeScript-en kereszül érik el a fent említett komponenseket.

## Megvalósítás

# Összefoglalás

# Ábrák jegyzéke

[1. ábra Gantt diagram a Microsoft Project 2013 programban 7](#_Toc419440676)

# Táblázatok jegyzéke

[1.1. táblázat. Példa táblázat feliratára 8](#_Toc396824940)

# Irodalomjegyzék

# Függelék

A függelék szövege.