Б.Сод-Од, Б.Гүндсамбуу, Б.Удвалцэцэг, Г.Цэнд-Аюуш

JavaScript фрэймворк

Улаанбаатар хот
2018 он

DDC

025.04

У2367

JavaScript

Зохиогчид: Б.Сод-Од
Б.Гүндсамбуу
Б.Удвалцэцэг
Г.Цэнд-Аюуш

Хавтасны дизайн: Б.Сод-Од

ISBN:

ӨМНӨХ ҮГ

Бид таны өмнө “JavaScript фреймворк” номоо өргөн барьж байна. Мэдээлэл технологийн хөгжүүлэлт, хэрэглээ улам нарийн болж хэрэглэгчдийн шаардлага өмнөх онуудаас өөр болж байна. Энэхүү ном нь ирээдүйтэй, өсөн нэмэгдэж яваа *JavaScript* -н тухай, яагаад ашиглах хэрэгтэй болон *JavaScript* дээр суурилсан чансаа өндөр байгаа фреймворкуудийг сонгон авч уншигч та бүхэнд хөгжүүлэх фреймворкоо сонгох мөн анхны эхлэлээ хэрхэн тавих талаар туслагч тань байх юм. Номны сэдвүүд *JavaScript* -н суурь мэдэгдэхүүнийг агуулаагүй учир вэб дизайны анхан шатны мэдэгдэхүүнтэй хүмүүст зориулсан.

Мөн их, дээд сургуулиудын вэб болон өгөгдлийн сан хөгжүүлэлт хичээлийн агуулгыг тренд болж байгаа технологиудтайгаа холбоход багахан нэмэр болох зорилготой. Цаашид тус фреймворкуудаас нэг нэгээр нь сонгон авч анхан шатнаас гүнзгий шат хүртэл хэрхэн хөгжүүлэлт хийх талаар номоо баяжуулан дараагийн номуудыг уншигч та бүхэнд хүргэх юм.

Энэхүү номыг бичихэд тусалсан хамт олон, гэр бүлдээ талархал илэрхийлье.

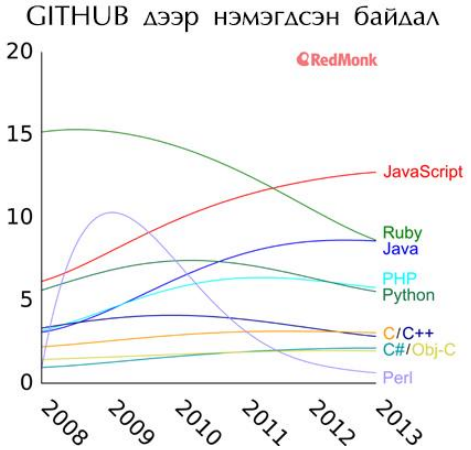
Монголын чадварлаг програмистууд маш олон болох болтугай!

Гарчиг

JAVASCRIPT -Н ТУХАЙ, АЖИЛЛАХ ОРЧИН БЭЛТГЭХ

1. JavaScript алдарт хүрсэн нь

JavaScript нь хамгийн анх 1995 оны 5 сард Нетскэйп компани өөрийн интернет хөтчид зориулсан гаргасан байдаг. Харин сүүлийн 5 жилд энэхүү хэл нь програм хангамж хөгжүүлэх гол түлхүүр хэл болсон байна. Доорх графикаас бид "*RedMonk*" сайтын цуглуулсан өгөгдлөөс *JavaScript* нь бусад програмчлалын акул хэлнүүдтэй хэрэглээгээрээ харьцуулагдсан байдлыг харж болно.



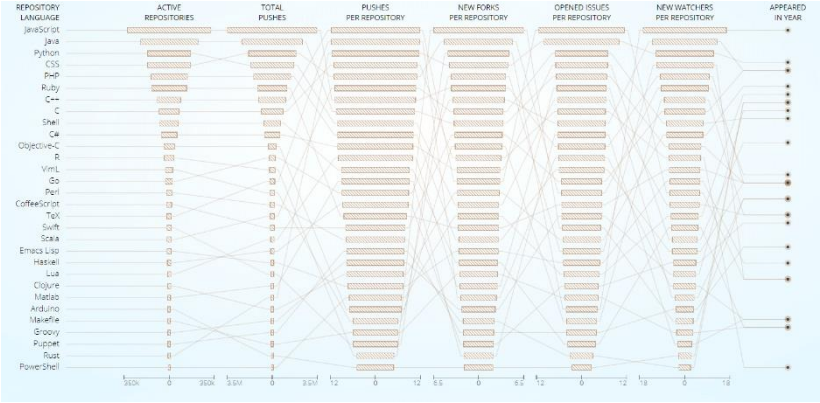
Зураг 1. https://redmonk.com/

Илүү дэлгэрэнгүй зураглал бас байна. "*Githut.info*" сайт дээрээс та бүхэн харж болно. Доорх графикт нээлттэй эхийн төслүүд дээр хийгдсэн судалгаагаар *JavaScript* нь програм хөгжүүлэлтийн гол тоглогч болсон тухайг илүү мэднэ.

1.1. JavaScript ашиглах шалтгаан

Шалтгаан 1: Энэ нь тархмал програм хангамжийн архитектурт зохимжтой.

Програм хангамж хөгжүүлэлтийн томоохон асуудал бол давхардал байдаг. Давхардал үүсэн нь нэг ажлыг олон удаа хийх шаардлагатай болгодог энэ нь хөгжүүлэлтийн зардлыг



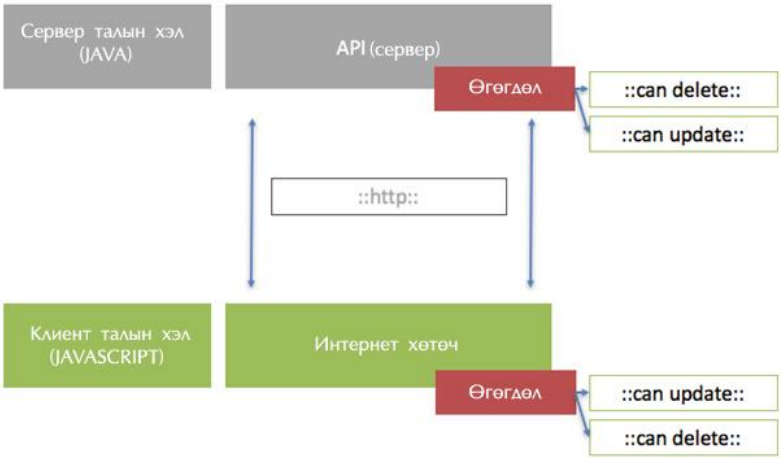
Зураг 2. http://githut.info/

шууд өсгөнө. Энэ нь бага биш нэлээдгүй зардал нэмэгддэг практик жишээнүүд байдаг. Харамсалтайн ихэнх тархмал архитектурууд хөгжүүлэлтийн үед давхардал үүсэхийн хэрээр зардал нэмж тухайн програмыг илүү хатуу ба засвар үйлчилгээ авахдаа муу болгож байна.

Жишээлбэл: Танд клиент, сервер програм байна. Нэг клиент (интернет хөтөч) бөгөөд энэ нь ямарваа нэг байгууллагын аюулгүй ажиллагааны чухал дүрмүүдийн жагсаалтыг хадгалах хэрэгтэй. Магадгүй нэмэх, засах, үйлдэл ч багтаж болно. Асуудал юу гэвэл та сервер дээр тэдгээр мэдээллийг хадгална. Техникийн багахан мэдлэгтэй хэн нэг нь л клиент талаар дамжуулан серверлүү хуурамч хүсэлт явуулж чадна. Ийм тохиолдолд бид асуудалд орно учир нь клиент тал нь *JavaScript* дээр бичигдсэн сервер программчлал нь жава эвсэл бусад программчлал дээр бичигдсэн байдаг тохиолдол нь элбэг.

Доорх зурагт улаан хүрээнд өгөгдлийн давхардал үүсэж болох нөхцөлийг харууллаа.

Энэ асуудлаас гарах арга зам нь бид дундын код хуваалцдаг арга техникийг хэрэглэг хэрэгтэй. Гэвч энэ нь хялбар биш учир нь бид өөр өөр программчлалын хэлнүүдийг хослуулсан хэрэглэдэг.



Зураг 3. 2 өөр програмчлалын дундах хил нь өгөгдлийн давхардал үүсгэдэг

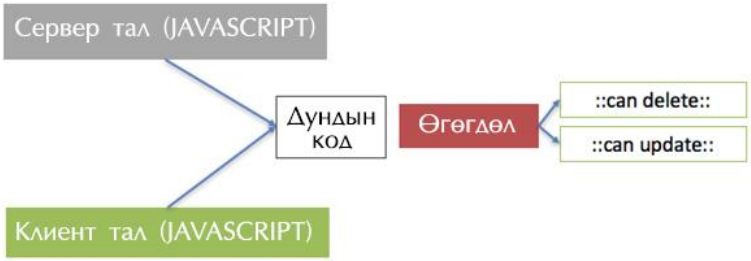
Харин одоо сервер талын хөгжүүлэлт хийдэг *JavaScript* гарч ирсэн.

JavaScript нь нэгэн төрлийн экосистем болж хурдацтай хөгжиж байна. Сервер талын хөгжүүлэлт хийдэг хүчирхэг багаж бол "*Google Chrome V8 Engine*" дээр суурилсан *NodeJS* юм. *Node* нь *JavaScript* нь сервер талд ажиллах боломж олгодог. Энэ нь та JavaScriptийг ашиглан бүтэн програмчлалыг хөгжүүлэх чадамж бүрдэнэ.

Томоохон компаниуд болох *Netflix, Walmart* нь сервер талын програмчлалын архитектураа (*ava,C#,PHP,Ruby* гэх хэлнүүд дүүр суурилсан байсныг *Node* болгон сольж эхэлсэн байна. Ингэснээр сервер болон клиент талтай тархмал програм хөгжүүлэхэд дундын код үүсгэх мөн дэд модуль хэсгүүдээ хуваан ашиглахад чөлөөтэй болох юм. Доорх зурагт дээр дурдсан асуудлыг *JavaScript* ашигласнаараа шийдэх боломжийг харууллаа.

Ингэснээр бид програм хангамжийн зохиомж гаргахад давхардлыг арилгаж дундын код хөгжүүлэлтийг

хэрэгжүүлснээр хөгжүүлэлтийн төвөгтэй байдлыг багасгаж засвар үйлчилгээ авахад илүү сайн болгож байна.



Шалтгаан 2: Хосолмол орчинг илүү дэмжинэ.

Энэ нь програмчлалаа нэг л удаа бичээд хаа сайгүй ажиллуулж болно гэдэг жава-гийн уриаг утгаар нь шийдэж өгч байгаа юм. Үүнийг бид кросс-платформ гэж нэрлэж сурсан бөгөөд энэ нь зах зээл хурдацтай өрсөлддөг болсон үед орчин бүрд тус тусад нь зориулан хөгжүүлэлт хийдэг байсан хэцүү явдлын аврал юм. Гэхдээ *JavaScript* нь гэнэт тийм ид шидтэй болоогүй бөгөөд 2007 оноос хойш Жеф Атвүүд гэх Америкийн програм хангамж хөгжүүлэгч эхлүүлсэн ажил юм. Өдгөө 11 жилийн дараах үр дүнгээс бид үр шимийн хүртэж байна.

"Ямар ч програм JavaScript дээр бичигдэж чадна, мөн бичигдэх болно" ~Жеф Атвүүд

JavaScript -г кросс-флатформ гэж нэрлэх нь араггүй юм. Учир нь десктоп, вэб, мобайл аль ч орчинг шийдэж чадна.

Мобайл төрөлд (*IOS, Android*) - *React Native* болон NativeScript нь мобайл орчинд зориулсан *JavaScript* -н томоохон хөгжүүлэлтийн платформ юм. React нь фэйсбүүк мобайл апп хөгжүүлэхдээ ашигладаг орчин юм.

Десктоп (*Mac, Windows*) - *NW.js* болон *Electron* нь вэб суурьтай боловч десктоп програмчлалд зориулсан платформууд юм.

Вэб (*All Browsers*) - *Angular, Meteor* гэх зэрэг олон тооны вэб

хөгжүүлэхэд зориулсан флатформууд, сангууд бий болсон. Энэ нь хөгжүүлэлт хийхэд хурдан бөгөөд уяан хатан тааламжтай байдаг.

Шалтгаан 3: Өргөтгөх, өсөн нэмэгдэх боломж

Сонирхолтой нь хэдэн жилийн өмнө *JavaScript* -н сул тал нь одоо хүчирхэг болж чадсан байна. Үүний бусад програмчлалаас илүү ялгаатай нь ганц урсгалтай програмчлал юм. *JavaScript* нь олон урсгалт боловсруулалт хийхгүй зөвхөн ганц урсгалт нь боловсруулалт хийнэ. Энэ нь илүү хөнгөн ажиллах боломжийг олгоно. Үүнийг хаалтгүй архитектур гэж нэрлэдэг. Энэ нь магадгүй анх сонсож байгаа програмистуудад кодоо үзэгдэлд суурилж хөгжүүлэх нь төвөгтэй санагдаж магадгүй л юм.

Олон усгалт програмчлал нь нөөцийг цаашдын боловсруулалтад нөөцлөн ашиглах бус нөөцийг шууд ил болгон гаргадаг. Жава энтерпрайз програмтай харьцуулахад хариу өгөх болон илгээх хугацаа нь бага байж чаддаг. Практик судалгаагаар *Node.js* нь *Java EE* гээс 20% хурдан ажиллагааг үзүүлдэг байна. Харин JavaScript нь урт хугацааны үйлчилгээний хүсэлтэд тохиромжтой биш байх тохиолдлууд байдаг. Жишээлбэл их хэмжээний өгөгдлийн хуулах, дүрс боловсруулах гэх мэт. Гэхдээ JavaScript нэгэнт эрчийг авсан...

2. JavaScript хэлний тухай үндсэн ойлголт

JavaScript нь компьютерийн динамик програмчлалын хэл бөгөөд вэб аппликэйшний хөгжүүлэлтэд хамгийн өргөн ашиглагдаж байна. Клиент талын *JavaScript* нь хэрэглэгчтэй харилцах, динамик хуудсыг бий болгох боломжийг бүрдүүлж өгдөг. *JavaScript* -г ашигласнаар серверийн харилцан үйлчлэл багасдаг. Тухайлбэл, хэрэглэгч сервер лүү хүсэлт илгээж эхлэхээс өмнө хэрэглэгчийн оролтыг баталгаажуулж болно.

JavaScript -г анх *Mocha* нэрийн дор *Netscape* компани хөгжүүлж эхлэсэн бөгөөд 1995 оны 9 сард нэрийг *LiveScript*

болгон өөрчилсөн боловч удалгүй *JavaScript* болгожээ. *ECMAScript* нь *JavaScript* -г стандартчилахад зориулж бүтээгдсэн скрипт хэлний тодорхойлолт бөгөөд жил тутам шинэчлэгдэж байдаг. Хамгийн сүүлийн үеийн стандарт багц нь *ECMAScript 2017*. Энэ нь *.js* өргөтгөлтэй бүх файлыг эмхэтгэнэ.

3. Суулгах програмчлалын хэрэгслүүд

JavaScript -тэй ажиллахын тулд дараах үндсэн бүрэлдэхүүн хэсгүүдээс тохирох програм хангамж, хэрэгслүүдийг өөрийн компьютерт суулгах хэрэгтэй.

3.1. Node Package Manager (NPM)

NPM бол “node package manager” гэсэн үгний товчлол бөгөөд *JavaScript* програмчлалын хэлний пакет менежер юм. Та *NPM* -ыг ашиглан үндсэн *JavaScript* -ийн сан, модуль, пакет болон бусад *JavaScript* хөгжүүлэгчдийн бүтээсэн сан, модуль, пакетыг суулгаж, ашиглах боломжтой. Энэ нь кодын дахин ашиглалтыг дэмжих бөгөөд өөрт хэрэгцээтэй пакетыг *https://npmjs.org* сайтаас авч ашиглана.

Мөн *NPM* нь *Angular 2* -той ажиллах орчин нөхцөлийг бүрдүүлж өгдөг. *Angular* аппликэйшны төсөлд хэд хэдэн хавтас болон файлууд байх ба *NPM* нь тэрхүү хавтас, файлуудыг үүсгэх, удирдах боломжоор хангадаг.

3.2. Node.js

Node.js бол *JavaScript* хэл дээр маш хүчирхэг, дахин ашиглах, өргөтгөгдөх боломжтой интернэт аппликэйшн бичихэд зориулсан орчин үеийн технологи юм. *Node.js* нь асинхрон оролт гаралт, бодит цаг хугацааны (*real-time*) ажиллагаа, кластер, грид (*grid computing*), үүлэн тооцоолол (*cloud computing)* зэрэг зүйлсийг ашиглах орчныг бий болгодог. Энэ нь тухайн код хурдан, үр ашигтай ажиллах боломжийг бүрдүүлдэг.

Дээр дурьдсан кластер, грид, үүлэн тооцооллыг хялбараар тайлбарлая. Сервер талаас харвал хэд хэдэн тооцоолуур нэг бодлого бодохыг кластер, олон кластер нэг бодлого бодохыг грид, олон грид нэг бодлого бодохыг үүл гэж ойлгож болно. Харин хэрэглээ талаас бол таны хатуу диск *Google* -ийн сервер дотор байхад, таны шуурхай санах ой өөр газар, мөн CPU чинь дэлхийн хаа нэгтээ гэх мэт. Тухайн үед танд ердөө дэлгэц, гар байхад л хангалттай болчих гээд байгаа юм.

Тэгэхээр энэ үүл гээч нь *Node.js* дээр хэрхэн хэрэгждэг вэ гэвэл та 10000 хэрэглэгч тус бүрийн 10 хүсэлтийг секунд тутамд авдаг гэж бодъё. Гэтэл энэ тоо ирэх жилийн өдийд 10 дахин нэмэгдэж болно. Хэрвээ та үүл гэдэг зүйлийг мэддэггүй, өөрөөр хэлбэл уламжлалт аргаар сэтгэвэл шууд л серверээ хүчирхэг болгох тухай бодол төрнө. Харин үүлэн дээр үүнийг өөр яг адил арван сервер хажууд нь залгаад, нэг холбогчоор (*middleware*) холбочихдог, ингээд л таны систем ажилладагаараа ажиллана. Үүнийг үүлэн тооцоолол гэх бөгөөд үүнийг Node.js төвөггүй хийнэ [1].

Нөгөөтэйгүүр *Node.js* -ын пакет болох *NPM* бол нээлттэй эхийн сангийн хамгийн том төлөөлөл юм. Тиймдээ ч *LinkedIn, Ebay, Yahoo!, Microsoft* гэх мэт маш олон гигантууд энэ системийг үр дүнтэй нэвтрүүлсэн.

Node.js технологи дээр суурилсан, уялдаатай ажилладаг авсаархан хирнээ хүчирхэг хэрэглээтэй зүйлс олон бий. Тухайлбал:

- *NPM - Node.js* -ийн пакет системийн удирдлага
- *Socket.io – WebSocket* -ийг *Node.js* дээр ажиллуулах ба боломжгүй бол *Ajax, Flash* зэргээр орлуулах боломжтой технологи
- *Express.js* - Вэб аппликэйшн фреймворк
- *LESS* - Динамик *CSS* фреймворк
- *Jade* - Динамик *HTML* фреймворк

- *Mongoose – MongoDB* -ийн *Node.js* сан

3.3. Angular CLI

Angular CLI (*Command line interface*) нь *Angular* -ийн програмуудыг үүсгэх, ашиглахад хялбар болгох зорилгоор хөгжүүлсэн хэрэгсэл юм. *Angular* -ийн ихэнх тохиргоонуудыг *Angular CLI* ашиглан хийх боломжтой бөгөөд уг ажлыг ихээхэн хялбарчилдаг.

3.3.1 Суурилуулах заавар

Angular CLI ашиглахын тулд хамгийн багадаа *NPM* 3.х.х болон *Node* -ийн 4.х.х хувилбараас хойшхийг суулгаж бэлтгэсэн байх шаардлагатай. Хэрэв дээрхээс бага хувилбар ашиглаж байгаа бол *NPM* болон *Node* -г заавал шинэчлэх хэрэгтэй.

Сервер дээр **`Local-web-server npm package`** ашиглана.
https://www.npmjs.com/package/local-web-server

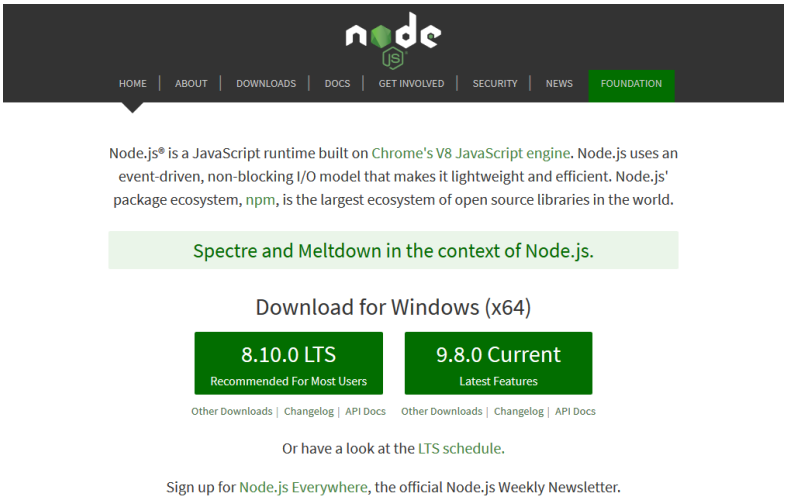
```
$ npm install -g local -web server
$ cd <your-app-folder>
$ ws
```

Үүний дараа өөрийн хөтөч дээр http:localhost:8181/ -ийг ачаална.

Клиент дээр *NPM* суулгаснаар таны компьютер дээр Angular аппликэйшнийг хөгжүүлэх орчин бүрдэнэ. *NPM* -г *https://nodejs.org/en/* хаягаар орж татан авч суулгана. *NPM* -д зориулсан албан ёсны сайт бол *https://www.npmjs.com/* юм.

Одоо *NPM* суулгах алхмуудыг дэлгэрэнгүй тайлбарлая.

- Татаж авсан суулгац файлыг ажиллуулж эхлүүлнэ. Эхний дэлгэц дээр *Next* товчийг дарна.
- Дараагийн дэлгэц дээр гарах лицензийг зөвшөөрвөл *Next* товчийг дарна.
- Дараагийн дэлгэц дээр байршуулах талбарыг сонгоод *Next* товчийг дарна.



Зураг 4. NPM татаж авах сайт

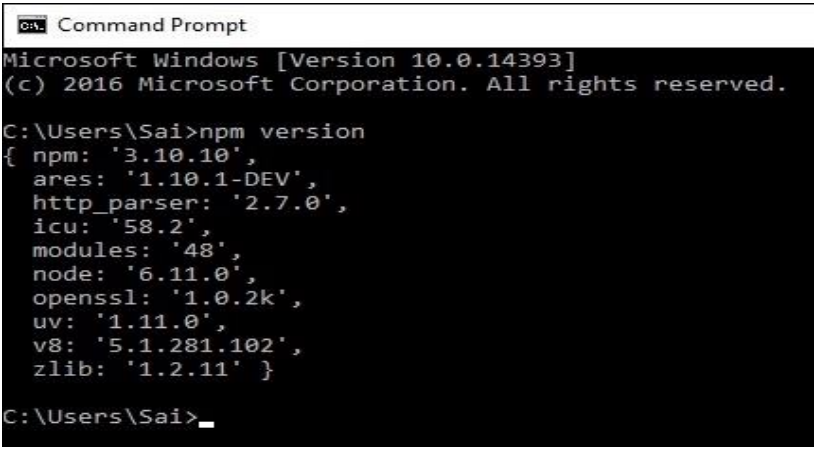
- Үүний дараа гарах дэлгэцээс шаардлагатай хэсгүүдийг сонгоод *Next* товч дарна. Та бүх бүрэлдэхүүн хэсгүүдийг анхдагч суулгацад хүлээн авах боломжтой.
- Дараагийн дэлгэцэн дээр *Install* товчийг дарна.
- Суулгац дууссаны дараа *Finish* товчийг дарна.
- Суулгацыг баталгаажуулахын тулд *CLI* мөрөнд npm хувилбар ажиллуулж болох бөгөөд дараах зурагт харуулсан шиг *npm version* -г харах боломжтой.

NPM суусан эсэхийг шалгахдаа *command prompt* цонхоо нээн **`npm -v`** гэж бичсэнээр компьютер дээр суусан *NPM* -н хувилбар гарч ирэх болно. Үүний тулд *command prompt* цонхыг нээж дараах командыг ашиглана.

```
npm install npm@latest -g
```

Хэрэв дээрх хувилбарууд компьютер дээр суусан бол *Angular CLI* суулгахын тулд төсөл үүсгэх хавтсанд очиж *command prompt* цонхоо нээж дараах командыг бичнэ.

```
npm install -g angular-cli
```



Зураг 5. npm version шалгах

Хэрэв *Angular CLI* суусан бол **`ng`** командыг ашиглах боломжтой болно.

3.4. Програмчлалын орчин (Editor)

JavaScript аппликэйшн хөгжүүлэх програмын эх кодыг бичих програмчлалын орчин хэрэгтэй. Ямар ч орчин (*Sublime text, Visual Studio Code, Atom* гэх мэт) дээр хийж болно.

- *Visual Studio Code* нь код бичихэд нүсэр биш, ойлгомжтой, шаардлагатай багаж хэрэгслүүдээр хангагдсан, бүх төрлийн үйлдлийн систем дээр зохицон ажиллах боломжтой, хөгжүүлэгчийн хийх үйлдлүүдийг хялбарчилсан орчин юм. Тухайлбал, *command prompt* -оо *Visual Studio Code* -оос удирдах (*Linux* үйлдлийн системтэй бол *termina l*-аа удирдах), шаардлагатай нэмэлт хэрэгслүүдийг суулгахад хялбар гэх мэт.
- *Sublime text* бол хувийн өмчийн бүх төрлийн үйлдлийн систем дээр ажиллах эх код засварлагч юм. Энэ орчин нь програмчлалын олон хэл, тэмдэглэгээний хэл

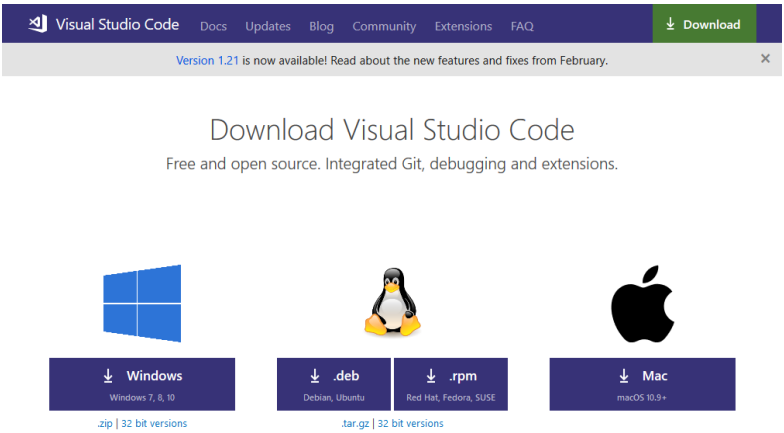
дэмжинэ.

Энд дурьдсан програмчлалын орчнуудаас жишээ болгож *Visual Studio Code* -г дэлгэрүүлж тайлбарлая. *Visual Studio Code* нь дараах онцлог талуудтай. Үүнд:

- *Visual Studio* хувилбартай харьцуулахад илүү хөнгөн;
- *Clojure*, *Java*, *Objective C* гэх мэт код бичихэд ашиглаж болно;
- *Git* өргөтгөлд бүтээгдсэн;
- *IntelliSense* горимд ажилладаг хөгжүүлэлтэд зориулсан олон нэмэлт өргөтгөлүүдтэй;

3.4.1 Visual Studio Code суулгах

Уг орчныг суулгахдаа https://code.visualstudio.com/download сайтад хандаж, өөрийн үйлдлийн системд тохирохыг татан авч суулгана.



Зураг 6. Visual Studio code татаж авах хаяг

Visual Studio Code суулгах алхмууд:

- Зураг 3 -т харуулсны дагуу тохирох файлыг татаж дууссаны дараа суулгалтын алхмуудыг дагана.
- Эхний дэлгэц дээрх *Next* товчийг дарна.
- Дараагийн дэлгэц дээр лицензийг зөвшөөрөөд *Next*

товчийг дарна.

- Дараагийн дэлгэц дээр суулгацын байршлыг сонгоод *Next* товчыг дарна.
- Програмын нэрийг сонгоод *Next* товчин дээр дарна.
- *Default* тохиргоог зөвшөөрч *Next* товчыг дарна.
- Дараагийн дэлгэц дээр *Install* товчийг дарна.
- Эцсийн дэлгэцэн дээр *Finish* товч дээр дарж *Visual Studio Code* -г эхлүүлнэ.

3.5. Өгөгдлийн сангийн шаардлага

Өгөгдлийн сан нь зохион байгуулалттайгаар хадгалсан өгөгдлийн цуглуулга юм. Өгөгдлийн сан нь талбар (*fields*), бичилтүүд (*records*) болон файлаас (*files*) бүрддэг. Талбар гэдэг нь багана бүхий мэдээлэл бөгөөд бичилт нь нэг мөрөнд байгаа нийт мэдээллийг хэлдэг. Нийт оруулсан мэдээллээ нэр өгч сануулан, файл болгодог. Өгөгдлийн сан нь мэдээлэл хайх, статистик мэдээ гаргаж авахад хялбар байдгаараа давуу талтай. Олон төрлийн өгөгдлийн сан удирдах системүүд байдаг бөгөөд энд *JavaScript* -д ашиглаж болох *MongoDB* өгөгдлийн сангийн тухай дэлгэрэнгүй тайлбарлая. Энэ санг их хэмжээний өгөгдөл, утасгүй хөдөлгөөнт системд ашиглавал илүү тохиромжтой.

3.5.1 MongoDB өгөгдлийн сангийн тухай үндсэн ойлголт

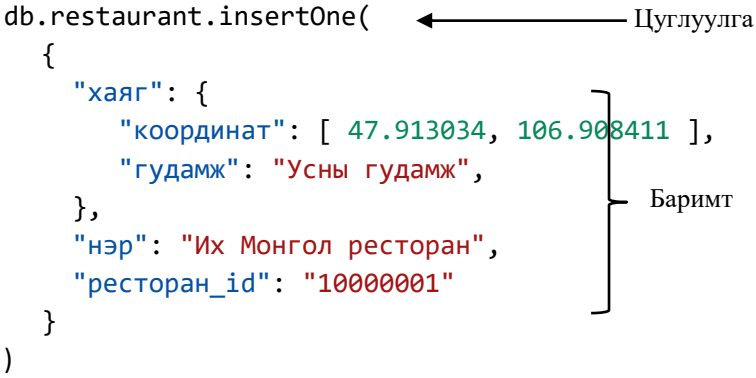
MongoDB бол нээлттэй эхийн төрөл бүрийн платформд тохирсон баримт (*document*) болон цуглуулгад (*collection*) суурилсан өгөгдлийн сангийн програм юм. *MongoDB* нь *C++, C* болон *JavaScript* хэл дээр бичигдсэн. Баримт гэдэг нь “түлхүүр, утга” хослолын олонлог бөгөөд *JSON* -той адил тодорхойлогддог. Динамик гэдэг нь ижил цуглуулгад байгаа баримт ижил талбар болон бүтэцтэй байх шаардлагагүй гэсэн үг юм.

MongoDB -н давуу талуудыг дурдвал:

- Тохируулахад хялбар;
- Нарийн төвөгтэй холбоосгүй;
- Хайлтыг гүн хийх боломжийг бүрдүүлж өгдөг;
- Холболт бага – нэг цуглуулга нь олон баримтыг агуулна;
- Шинэчлэл хурдан хийдэг;
- Уламжлалт өгөгдлийн сангийн боловсруулалтаас бараг 100 дахин хурдан;
- Нэг объектын бүтцээр илэрхийлэгдэнэ;

Баримт нь динамик схемтэй байдаг. Динамик схем гэдэг нь ижил цуглуулгатай баримтууд ижил талбар эсвэл ижил бүтэцтэй байх албагүй. Цуглуулгын баримт дахь нийтлэг талбарууд янз бүрийн өгөгдлийг агуулж болно.

Тухайлбал, дараах жишээнд *db.user.insertOne* мөр нь цуглуулгын нэг хэсгийг илэрхийлж байгаа ба хэрэглэгчийн мэдээллийг багтаасан байна. Үүний хэрэглэгчийн нэр, нас дугаар нь баримтыг илэрхийлнэ. *MongoDB* өгөгдлийн сан нь өгөгдлийг *JSON* загвараар хадгалдаг.



Зураг 7. MongoDB -д өгөгдөл хадгалах загвар

MongoDb өгөгдлийн санг уламжлалт өгөгдлийн сан удирдах системтэй (*RDBMS – Relational Database management system*) харьцуулбал:

RDBMS	MongoDB
Хүснэгт (Table)	Цуглуулга (Collection)
Мөр (Tuple/Row)	Баримт (Document)
Багана (Column)	Талбар (Field)
Хүснэгт нэгтгэх (Table join)	Эмбэддэд баримт (Embedded document)
Анхдагч түлхүүр (Primary key)	Анхдагч түлхүүр
Сервер болон клиент	
Mysqld/Oracle	mongod
Mysql/sqlplus	mongo

ANGULAR 2

4. Angular 2

AngularJS (ихэвчлэн *Angular.js* эсвэл *AngularJS* гэж нэрлэдэг) анх 2009 онд *Google* -ээс боловсруулсан нээлттэй эхийн *front-end* талд ашиглагдах *JavaScript* фреймворк юм. *Angular* нэрний үүслийн тухайд *HTML* тагуудыг илэрхийлдэг <> буюу хаалт тэмдэгт (*angle brackets*) -ээс гаралтай. Үндсэндээ *HTML* бичиглэлийг аппликэйшн хөгжүүлэхэд боломжтой болгох зорилготой. Энэ нь өгөгдлийн урсгал болон *HTML* загвар гэсэн ойлголтуудыг цогцоор агуулсан. *AngularJS* -н хөгжүүлэлт тасралтгүй хийгдсээр байгаа ба 2014 онд хувилбар 2, 2017 оны гуравдугаар сард хувилбар 4 нь гарсан. Хувилбар 3 -г шууд алгасаж 4 -р хувилбарыг гаргасан шалтгаан нь тухайн үед *Angular* -н үндсэн сан болох *core, compiler, compiler-CLI, http* нь адил зарчмаар v2.3.0 дугаарлагдсан байтал *route* сан нь v3.3.0 хувилбар дээр байв. Иймд хөгжүүлэгч баг цаашдын хөгжүүлэлтэд ямар нэг эргэлзээ үүсгэхгүйн тулд хувилбар 2 -оос шууд 4 -рүү шилжсэн байна.

2014 онд гарсан *Angular 2* шинэчлэлт компонентыг хөгжүүлэх боломжийг олгосноороо давуу тал болсон. *Angular 2* нь *Angular* сангийн бүрэн хэмжээний засвар төдийгүй *Angular 1* -ийн програмуудтай нийцэж ажилладаггүй. Энэ нь хөгжүүлэгчдийн санааг зовоож байсан ч *Angular 2* -т тусгаж өгсөн шинэчлэлтүүд нь 2009 онд байгаагүй олон боломжийг бий болгосон юм.

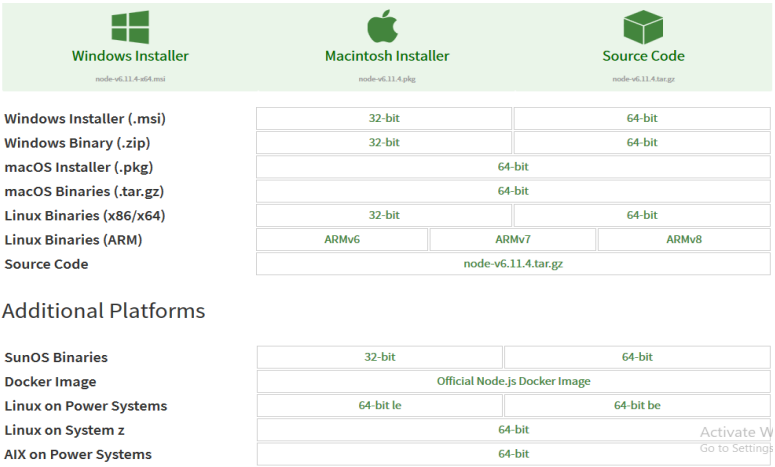
Вэб хуудсыг үүсгэх үед өөрийн бичсэн код нэмэлт сангуудтай давхцах бий гэж санаа зоволтгүйгээр өөрийн хүссэнээр ажиллах компонентыг үүсгэх боломжийг олгосон нь *Angular* -н гол давуу тал юм. *Angular* нь *front-end* боловч хөгжүүлэлтийн уламжлалт арга барилаас өөр. Учир нь *front-end* вэб хөгжүүлэлтийг бусад хөгжүүлэлтийн орчин шиг болгож ашиглах, хүссэн компонентуудыг нэгтгэн нэг цогц болгох боломжийг бидэнд олгоно. Маш энгийнээр бол

AngularJS програм нь *HTML*, *CSS*, *JavaScript* болон *JavaScript* -г хөрвүүлдэг бусад скрипт хэлийг ашиглан клиент аппликэйшн бүтээх боломжийг бидэнд олгодог. Хөгжүүлэгч нь контроллерийнхоо утгыг удирдана. Харин *AngularJS* нь өөрчлөгдсөн утгуудыг тусгахын тулд *DOM* (*Document object model*) -г шинэчилдэг.

AngularJS -г *Wolfram Alpha*, *NBC*, *Walgreens*, *Intel*, *Sprint*, *ABC News* болон өөр бусад 12,000 орчим сайт дээр ашиглаж байна. Мөн *AngularJS* 2018 оны байдлаар *GitHub* дээрх шилдэг 100 төсөлд багтсаар байна.

4.1. Үйлдлийн системийн шаардлага

Вэбд суурилсан учир үйлдлийн системээс хамааралгүй. *Angular 2* -г ашиглахын тулд таны компьютер хамгийн багадаа *Node* 4.x.x болон *NPM* -ийн 3.x.x хувилбар суусан байх шаардлагатай. Харин техник хангамжийн талаас шаардагдах үзүүлэлтийг дараах зурагт харуулав.



Зураг 8. Үйлдлийн системийн шаардлага

4.2. Суулгах програмчлалын хэрэгслүүд

Angular 2 -той ажиллахын тулд дараах үндсэн бүрэлдэхүүн хэсгүүдээс тохирох програм хангамжуудыг суулгах хэрэгтэй.

- Node Package Manager (NPM)
- Node.js
- Angular CLI

4.3. Програмчлалын хэлний шаардлага

4.3.1 ECMAScript 6

ECMAScript нь *ECMA* стандартаар гаргасан скрипт хэл ба *JavaScript* нь клиент талын вэб програмуудын *ECMAScript* -н хамгийн өргөн хэрэглэгддэг хэрэгжүүлэлтийн нэг юм. Анх 1997 онд зарлагдсан ба хэдхэн жилийн дотор хурдацтай хөгжиж, шинэчлэгдсээр байна. Сүүлийн хувилбар болох ES6 нь цогц скрипт програм бичихэд чиглэгдэж, бичиглэлийн сайжруулалтыг анхаарсан байна. Энэ шинэчлэлтийг бүх вэб хөтөч дэмжихгүй байгаа ч *ECMAScript 6* нь *JavaScript* -н ирээдүй гэгдсээр байна.

4.3.2 TypeScript 6

TypeScript бол томоохон програмуудыг хөгжүүлэхэд зориулагдсан *JavaScript* юм. *TypeScript* нь үнэгүй, нээлттэй эхийн програмчлалын хэл бөгөөд Microsoft компаний *Anders Hejlsberg* (*C#* програмчлалын хэлийг зохиосон) програм хангамжийн инженер хөгжүүлсэн. Үүний хамгийн чухал шинэчлэлт бол *JavaScript* -д өгөгдлийн төрлийн боломжийг илүү нэмсэн. *TypeScript* файл нь эх кодыг *JavaScript* файл руу хөрвүүлдэг. Ингэснээр хөгжүүлэгчид *TypeScript* -н функцийг ашиглах боломж олгоно. Мөн *JavaScript* рүү хөрвүүлэгдсэн *TypeScript* -г хөтөч ойлгох боломжийг олгодог.

Angular 2 нь өөрөө *TypeScript* -ээр бичдэг. Гэхдээ *TypeScript* мэдэхгүй гэж санаа зовох хэрэггүй. *JavaScript* болон объект хандлагат програмчлалын суурь мэдлэгтэй бол *TypeScript* -н үндсэн суурийг зөвөөр ойлгох боломжтой.

TypeScript -н зарим нэг онцлогуудыг дурдая. Үүнд:

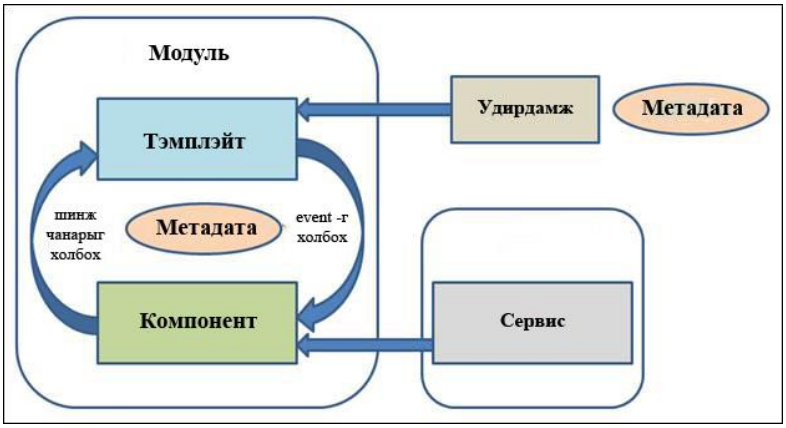
- *Type inference* – энэ нь програмчлалын хэл дээр илэрхийгдэх өгөгдлийн төрлийг автоматаар гаргалгаа хийнэ гэсэн үг.
- *Type annotations* – арга болон функцуудын оролт гаралтыг тодорхойлдог.
- *Interface* – харилцан хамааралгүй объектуудын нэг нөгөөтэйгээ харилцах нийтлэг ойлголт юм.
- *Enumerated type* – элемент, гишүүд болон тоолуур зэргийн утгуудыг агуулсан өгөгдлийн төрөл юм.
- *Mixin* – бусад классуудын эцэг классгүйгээр бусад классуудыг ашиглах аргуудыг агуулсан класс юм.
- *Namespace* – төрөл бүрийн объектуудыг зохион байгуулахад ашигладаг тэмдэгтүүдийн багц юм. Тэдгээр объектуудыг нэрээр нь тодорхойлж болох юм.
- *Tuple* – элементүүдийн төгсгөлтэй захиалсан жагсаалт юм.

4.4. AngularJS -н боломж

4.4.1 Dependency injection

Dependency injection нь компонентуудын хоорондын холбоо хамаарлыг өөрчлөх замаар нягт холбоог багасгахыг оролддог програм хангамжийн зохиомжийн загвар юм. Компонентуудын нягт холбоо хамаарал нь компонентыг тест хийх болон дэбаг хийхэд хүндрэл учруулдаг.

Бүрэлдэхүүн хэсгүүд – *Angular* -ийн өмнөх хувилбар нь *controller* -уудад түлхүү анхаарч байсан бол одоо бүрдэл хэсгүүдийн *controller* руу илүүтэй анхаарал хандуулж байна. Бүрэлдэхүүн хэсгүүд нь програмуудыг олон модуль болгон хөгжүүлдэг бөгөөд тухайн програмыг цаг тухай бүрт нь сайжруулахад дөхөмтэй.



Зураг 9. Angular -н архитектур

- Модуль (*module*) - Энэ нь програмыг логик хэсгүүдэд задлахад хэрэглэгддэг. Код буюу модуль бүр нэг үүрэг гүйцэтгэх зориулалттай.
- Компонент (*component*) - Энэ нь модулиудыг нэгтгэж ашиглах боломжийг олгоно.
- Тэмплэйт (*template*) - Энэ нь *AngularJS* програмын харагдацыг тодорхойлоход хэрэглэнэ.
- Метадата (*metadata*) - Энэ нь *AngularJS* классд өгөгдөл нэмэхэд хэрэглэнэ.
- Сервис (*service*) - Энэ нь програмд бүхэлд нь хуваалцаж болох компонентийг үүсгэхэд хэрэглэгддэг. Сервис нь програмын өөр өөр хэсгүүдэд хуваарь болох кодын багц юм. Жишээлбэл, та өгөгдлийн сангаас өгөгдөл цуглуулан *shared service* байдлаар олон төрлийн програм дээр ашиглах боломжтой.

4.5. AngularJS давуу тал

Вэбийг *Single Page* байдлаар үр ашигтай хөгжүүлэх боломж олгодог, TypeScript ашиглан бичдэг тул Объект Хандалтат Технологи хэрэглэх бүрэн боломжтой. *Angular CLI* суулгаад програм хөгжүүлэх үндсэн орчинг бэлдэж өгдөг. Үндсэндээ програмын үндсэн бүтэц буюу араг ясыг

гаргаж өгдөг. *HTML* хуудсыг хязгаарлан жижиг хэсэгт хуваах боломж олгодог. Ингэснээр хэрэглэгчид цэгцтэй хариу үйлдэл үзүүлэх боломжтой.

- Нэгжийн тест хийх боломж: Бага кодоор илүү үр дүнд хүрэх, *Android, IOS* гар утас болон таблет гэх мэт төхөөрөмжүүд дээр ч ажиллах боломжтой.
- *Google* -н хөгжүүлэлт: *AngularJS* -ийг *Google* -ийн инженерүүд тусгайлан боловсруулсан. Үүний зэрэгцээ хөгжүүлэлтийн явцад тулгарсан аливаа бэрхшээлийг даван туулахад туслах зөвлөх инженерийн баг ажиллаж байдаг. Ингэснээр суралцаж буй хүмүүс мэдэхийг хүссэн зүйлсээ асуух боломжтой.
- Маш сайн *MVC*: Ихэнх технологууд програмыг олон MVC бүрэлдэхүүн хэсгүүдэд хуваахыг шаарддаг. Үүний дараа программист дахин нэгтгэхийн тулд код бичих хэрэгтэй болдог. *AngularJS* нь автоматаар нэгтгэдэг. Энэ нь цаг хугацааг маш их хэмнэдэг.
- Бүрэн цогц: *AngularJS* нь *front-end* хөгжүүлэлтийг түргэтгэх цогц шийдэл юм. Үүнд ямар ч нэмэлт *plugin* эсвэл *framework* хэрэггүй.
- Ухаалаг: *AngularJS HTML* суурьтай хэлийг ашиглахад илүү хялбар. Үүнээс гадна, дахин зохион байгуулахад илүү амар байдаг.

4.6. AngularJS сул тал

Ямар ч платформд давуу талуудтай зэрэгцэн сул тал оршдог билээ. *AngularJS* -д ч мөн адил. *AngularJS* -г ашиглахад илрэх зарим сул талууд:

- Зөвхөн *AngularJS* дээр бичигдсэн програм нь аюулгүй байдал тал дээр дутагдалтай.
- Хэрвээ програм хэрэглэгчийн ашигладаг хөтөч нь *JavaScript* дэмждэггүй бол програм ажиллахгүй.
- *Angular* дээр бичигдсэн програмыг *Angular 2*

дэмждэггүй.

- *Angular* програм хөгжүүлж эхлэхийн тулд хөгжүүлэгч *TypeScript* мэддэг байх хэрэгтэй.
- Жижиг вэб аппликэйшн хөгжүүлэхэд зохимжгүй. Таны вэбийн хэмжээ 2мб болох байтал *Angular* төсөл үүсгэхэд *node_modules* хавтас гэхэд 200мб файл татагддаг.
- *AngularJS* -н тусламжтай олон *NG* програмыг нэг хуудсан дээр үүсгэх боломжгүй. Нэрүүд зөрчилдөж болзошгүй.
- *Angular* нь том бөгөөд илүү төвөгтэй. "*Hello world"* хэвлэхэд бусад хэлэнд илүү хялбар байдаг бол *Angular* -н хувьд арай төвөгтэй байдаг нь ажиглагдана.

4.7. Бусад ижил төстэй технологи, тэдгээртэй харьцуулсан судалгаа

Angular.js болон *React.js* нь *front-end* хөгжүүлэлтэнд түгээмэл ашиглагддаг технологиуд юм.

AngularJs, React.js, Ember.js харьцуулсан судалгаа

Фреймворк	**AngularJS**	**ReactJS**	**Ember.js**
Юу болох	*Superheroic JavaScript MVW Framework*	Хэрэглэгчийн интерфэйсийг бүтээхэд зориулсан *JavaScript* сан	Томоохон вэб програмыг бий болгох фреймворк
Түүх	2009 онд *Miško Hevery* үүсгэн байгуулагдсан	2013 онд *Jordan Walke* нээлттэй эхтэйгээр бүтээсэн	Эхлээд *Yehuda Katz* нь *SproutCore* нэртэй 2007 онд байгуулсан бөгөөд *Facebook* худалдан авснаар 2011 онд *EmberJS* гэж нэрлэсэн.

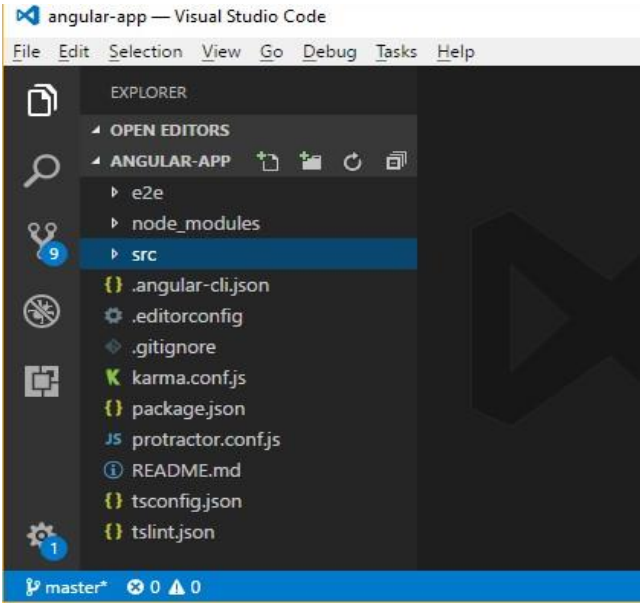
Вэбсайт	https://angularjs.org/	https://reactjs.net/	http://emberjs.com/
Github	https://github.com/angular/angular.js	https://github.com/facebook/react	https://github.com/emberjs/ember.js
Алдааны тайлан *(bug)*	https://github.com/angular/angular.js/issues	https://github.com/facebook/react/issues	
Лиценз	*MIT*	*MIT*	*BSD-3-Clause*
Энэ технологийг ашигладаг алдартай вэбсайтууд	*Youtube, Vevo, Freelancer, Istockphoto, Weather, Sky Store*	*Facebook, Instagram, Khan Academy, New York Times, Airbnb, Flipkart, Sony Lifelog*	*Apple Music, Yahoo!, LinkedIn, TinderBox, Netflix, Groupon*
Зориулалт	Өндөр идэвхтэй, интерактив вэб програмууд бүтээх	Өгөгдөл нь байнга өөрчлөгдөж байдаг том вэб програмууд	Dynamic SPAs
Хувилбар	2.0	15.4.0	
Компани	*Google*	*Facebook*	
Хэл	*TypeScript*	*JSX*	
Кодын загвар	*JS into HTML*	*JavaScript Centric*	
DOM	*Regular DOM*	*Virtual DOM*	
Debugging	*Good JS/Good HTML*	*Good JS/Bad HTML*	
Унах тохиолдол	*Runtime*	*Compile-Time*	
Холболт	*2 way*	*Uni-*	

		directional	
Темплэйт	*TypeScript файл*	*JSX файл*	
Компонент загвар	Хүчтэй	Дундаж	
MVC	*Дэмжинэ*	*View* -г ашиглана.	

4.8. Хавтасны зохион байгуулалт

Angular 2 програм үүсэхэд хэд хэдэн хавтаснууд үүсдэг. Эдгээр хавтас таны програмд хэрэгтэй програм, эх код болон зарим нэг тестийн хэрэгслийг агуулсан байдаг. *Angular 2* аппликэйшнд агуулагдах хэд хэдэн нийтлэг хавтас байдаг. Үүнд:

- *src* - аппликэйшны эх кодыг агуулах үндсэн хавтас
- *node_modules* - хэрэгцээт санг агуулсан хавтас
- *e2e* - тестийн хавтас



Зураг 10. AngularJS төслийн хавтас Visual Studio Code дээр

Шинээр үүсгэсэн *Angular* төслийн хавтасны бүтэц нь дараах байдлаар харагдана.

4.9. Файлын зохион байгуулалт

Үндсэн програм шаардлагатай хэд хэдэн *Git* болон *GitHub* файлуудыг агуулдаг. Эдгээр нь *.gitignore* болон *README.md* файлууд юм.

- *.gitignore* файл бол *Git version control* -оос ирсэн *GitHub* болон *Git* -д зориулсан стандарт файл юм. Та репозиторыг шалгахын тулд файл, хавтсыг тодорхойлж болно. Тухайлбал:
  - */dist: distribution* хавтасны файлыг оруулахгүй байх
  - *.map:* газрын зургийн файлыг оруулахгүй байх
  - */node_modules: Angular* -н эсвэл бусад санг оруулахгүй байх
- *README.md GitHub* төсөл нь *README.md* файлыг агуулж байдаг. *Angular CLI* шинэ төсөл үүсгэх үед хэрэглэгчид зориулсан *readme* файлыг агуулдаг. Хэрвээ *GitHub* ашиглахгүй бол энэхүү файлыг тоохгүй байж болно.
- *.editorconftg* файл нь *editor* -т тохиргоо хийх боломжийг олгоно. Энэхүү файлд *indent_style* болон *indent_size* гэх мэт тохиргоог хийж болно. Зарим нэг *editor* -ууд энэхүү файлыг дэмжиж ажиллахгүй ба хүсвэл өөрийн сонгосон *editor* -т нэмэлт *plug-in* суулган ажиллуулж болно.
- *angular-cli.json* файл нь *angular CLI* хэрэгслийн ажиллагааг тохируулах боломжтой. Энд програм, төсөл, үндсэн тохиргоо гэх мэт олон хэсгүүд байна. Програм хэсэгт тохируулах боломжтой ихэнх тохиргоонууд байдаг. Мэдвэл зохистой тохиргоог дурдвал:
  - *apps/root: root* файл хаана байрлах эсэхийг

- *apps/outDir: dist* файлууд хаана байрлах эсэхийг
- *apps/assets: distribution* -д байх ёстой файл болон хавтсууд
- *apps/styles:* аппликэйшнд ашиглагдах css -үүд хаана байрлах
- *apps/scripts: JavaScript* файлууд хаана байрлах зэргийг тохируулж болно.

4.10. Ашиглагдах функц, үйлдлүүд

4.10.1 Компонент

Angular -н нэг том давуу тал бөгөөд энэ нь хуудаст аппликэйшн үүсгэх боломжийг олгодог. Өөрөөр хэлбэл *HTML* кодыг тодорхой хэсгүүдэд хуваана. Тэрхүү хуваагдсан хэсгүүдийг бүрэлдэхүүн хэсэг буюу компонент гэж нэрлэнэ. Компонентод тухайн хэсгийн *HTML* код, загвар, арын боловсруулалт хийх код гэх мэт тухайн хэсэгтэй холбоотой бүхий л зүйл хамаарна. Компонентын жишээг дараах зурагт харуулья.

```
import {Component, OnInit}  from '@angular/core';
@Component({
    selector: 'app-react-js',
    templateUrl: './react-js.component.html',
    styleUrls: ['./react-js.component.css']
})
export class ReactJsComponent implements OnInit {
    constructor(){}
    ngOnInit(){
    }
}
```

Зураг 11. AngularJS компонент

4.10.2 Директив (Angular Directive)

***ng** угтвартай тусгай атрибутууд юм. Директивийн утга ямар нэг *JavaScript* илэрхийлэл байна. Директивийн үүрэг бол

өөрийн утга өөрчлөгдөх үед *DOM* -д тохирох нөлөөг үзүүлэх юм. Тухайлбал, дараах жишээнд буй ***ngIf** директив нь *apples* -н утгаас хамааран <*p*> тагийн утгыг харуулах эсэхийг шийдэж байна.

```
<p *ngIf = "apples.length > 3">Энд маш олон алим
байна!</p>
```

*Зураг 12. AngularJS -д ***ngIf** директив ашигласан жишээ*

4.10.3 Өгөгдлийг дүрслэн үзүүлэх

Компонентийн шинж чанаруудыг *HTML* загвараар дүрслэн үзүүлж болно. Компонентийн шинж чанарыг харуулах хамгийн энгийн арга бол тухайн шинж чанарын нэрийг интерполяциар (*angular interpolation*) холбох юм. Интерполяцийг ашиглан шинж чанарыг харуулахдаа давхар угалзан хаалтанд шинж чанарын нэрийг бичдэг.

```
import {Component}  from '@angular/core';
@Component({
    selector: 'my-app',
    template: `
        <h1>{{title}}</h1>
        <h2>Миний дуртай ном бол: {{myBook}}
</h2>`
})
export class AppComponent {
    title = 'Ном';
    myBook = 'Харри Поттер';
}
```

Зураг 13. AngularJS -д өгөгдөл дүрслэх

Өгөгдлийг интерполяциар дүрсэлснээр *component* -н тухайн шинж чанарын утга өөрчлөгдөхөд вэб хуудсыг дахин ачаалалгүйгээр загварын утга шинэчлэгдэх болно.

4.10.4 Байгуулагч функц

Component -д ямар нэгэн хувьсагч хэрэгтэй, мөн тэрхүү хувьсагчид анхны утга олгох хэрэгтэй бол байгуулагч функцийг ашиглаж болно.

```
export class AppComponent {
    title: string;
    myBook: string;
constructor(){
        this.title = 'Ном';
        this.myBook = 'Харри Поттер';
    }
}
```

Зураг 14. AngularJS -д байгуулагч функц зарлах

4.10.5 Массив

Массивыг дэлгэцэнд дүрслэхдээ ***ngFor** -г ашиглана. Дараах зурагт массив зарлах жишээг харуулав.

```
export class AppComponent {
    title = 'Ном';
    books = ['Харри Поттер', 'Зуун жилийн
ганцаардал', 'Да Винчийн код'];
    myBook = this.books[1]; //'Зуун жилийн
ганцаардал'
}
```

Зураг 15. AngularJS -д массив зарлах

***ngFor** нь **<ul>** болон **<li>** тагийг ашиглан элеменийг давтан харуулна. ***ngFor** -г ашиглах жишээг дараах зурагт харуулав.

```
<ul>
    <li *ngFor = "let book of books">
        {{book}}
    </li>
</ul>
```

*Зураг 16. AngularJS 2-д ***ngFor** ашиглах*

Дээрх тохиолдол зөвхөн массивыг дүрсэлж байна. ***ngFor** нь давтан харуулах боломжтой бүхнийг дүрслэн харуулах боломжтой.

4.10.6 Нөхцөл шалгах

AngularJS -д нөхцөл шалгахдаа ***ngIf** -г ашиглана. Дараах жишээнд ***ngIf** -г ашиглан дэлгэцэнд өгөгдлийг дүрслэх жишээг харуулж байна.

```
<p *ngIf = "books.length > 10">Энд маш олон ном
байна!</p>
```

*Зураг 17. AngularJS -д ***ngIf** ашиглах*

4.10.7 Өгөгдлийн урсгал болон оролт, гаралтын үйлдлүүд

Хэрэглэгчийн оруулсан өгөгдлөөс шалтгаалж *DOM* -н үйлдлүүд өдөөгдөж байдаг. *DOM* -н үйлдлийг холбохын тулд тухайн үйлдлийн нэрийг хаалтанд бичиж үндсэн загварын дагуу бичнэ. Тухайлбал, тухайн товч дарагдах үйлдлийг дараах байдлаар дүрсэлж болно.

```
<button (click) = "onClickMe()">Энд дар!</button>
```

Зураг 18. AngularJS дээр товч дарагдах үйлдэл

Ямар нэг үйлдэл хийхийг загварт дүрслэхдээ дуудах функцийн нэрийг мэдэж байх ёстой. Дуудах функц нь тухайн компонентод тодорхойлогдсон байх шаардлагатай.

Загварт байгаа товчийг дарснаар *ClickmeComponent* -н *onClickMe()* функц дуудагдана.

```
@Component({
    selector: 'click-me',
    template: `
    <button (click) = "onClickMe()">Энд
дар!</button>
    {{clickMessage}}`
})
export class ClickComponent {
    clickMessage = '';
    onClickMe()
    {
        this.clickMessage = 'Энэ бол миний дуртай
ном!';
    }
}
```

Зураг 19. AngularJS дээр товч дарагдах үйлдэл 2

DOM -н үйлдлүүд өөртөө мэдээлэл агуулсан байдаг. Энэхүү мэдээллийг компонент дэх функцэд ашиглах боломжтой. Товч дарагдах бүр хэрэглэгчийн өгөгдлийг авахын тулд *inputbox* -н *keyup* үйлдлийг хэрхэн холбохыг жишээ болгон харуулъя.

```
@Component({
    selector: 'app-key-up',
    template: `
      <input (keyup)="onKey($event)">
      <p>{{values}}</p>
    `
  })
  export class KeyUpComponent_v1 {
```

```
    values = '';
    onKey(event: any) {
      this.values += event.target.value + ' | ';
    }
  }
```

Зураг 20. AngularJS -д үйлдлийн мэдээлэл дамжуулах

Үйлдлийн объектын шинж чанар нь *DOM* -ийн үйлдлийн шинж чанараас хамааран өөр өөр байдаг. Бүх стандарт *DOM* -ийн үйлдлийн объектууд *target* шинж чанартай байх тул тухайн үйлдлийн утгуудыг агуулсан байна.

4.10.8 Замчлал

Нэг хуудаст аппликэйшн хөгжүүлэхийн тулд замчлал гэдэг зүйл зайлшгүй хэрэгтэй. Энэ нь компонент хоорондын шилжүүлэлтийг хариуцдаг бөгөөд хэрэглэгчдэд хуудас солихгүйгээр өөр, өөр зүйлсийг харуулах боломжийг бүрдүүлдэг. Замчлалыг хийхдээ дараах байдлаар хийнэ.

```
export const routes: Routes = [
  { path: 'angular-js', component:
AngularJsComponent },
  { path: 'react-js', component: ReactJsComponent
},
  { path: 'dashboard', component:
DashboardComponent },
  { path: 'detail/:id', component:
HeroDetailComponent },
  { path: 'books', component: BooksComponent }
];
export const routes: ModuleWithProviders =
RouterModule.forRoot(routes)
```

Зураг 21. AngularJS -д замчлал хийх

4.11. Технологийн хэрэглээ

AngularJS нь *Firebase* -г *back-end service* болгон хамтран ажиллах боломжтой. Firebase -н сервисүүдийг тайлбарлая.

- *Firebase Auth* – Энэ нь зөвхөн клиент талын код ашиглан хэрэглэгчдийг баталгаажуулах үйлчилгээ юм. Энэ сервис нь *Facebook, GitHub, Twitter, Google* гэх мэт хэрэглэгчийн бүртгэлээр таны аппликэйшнд Нэвтрэх боломжийг олгоно. Мөн таны аппликэйшнаар дамжуулан имэйл болон нууц үгээр бүртгүүлэн нэвтрэх боломжтой.
- *Realtime Database* - Энэхүү сервис нь аппликэйшн хөгжүүлэгчдэд аппликэйшны өгөгдлийг хэрэглэгчдэд синхрон үзүүлэх боломжийг олгодог. Мөн *Firebase - Cloud* дээр хадгалах боломжийг олгодог. Мөн өгөгдлийг бусад төрлийн аппликэйшн дундаа ашиглах боломжийг олгоно.
- *Firebase Storage* - Сүлжээний чанараас үл хамааран *Firebase* аппликэйшнуудад аюулгүй файлын байршуулалт, татаж авах боломжийг олгодог. Хөгжүүлэгч нь зураг, аудио, видео болон бусад хэрэглэгчийн үүсгэсэн контентыг хадгалах боломжтой. *Google Cloud Storage* нь *Firebase Storage* нь дэмждэг.
- *Firebase Hosting* - 2014 оны 5-р сарын 13-нд эхлүүлсэн статик, динамик вэб хостинг үйлчилгээ юм. Энэ нь *CSS, HTML, JavaScript* болон бусад файлыг статик файлаар дэмждэг бөгөөд *Cloud Functions* -ээр дамжин динамик *Node.js* дэмжлэгийг үзүүлдэг. Энэ үйлчилгээ *HTTP Secure (HTTPS)* болон *Secure Sockets Layer* шифрлэлт *(SSL)* -аар дамжуулан контент дамжуулах сүлжээ *(CDN)* -ээр файлуудыг дамжуулдаг.

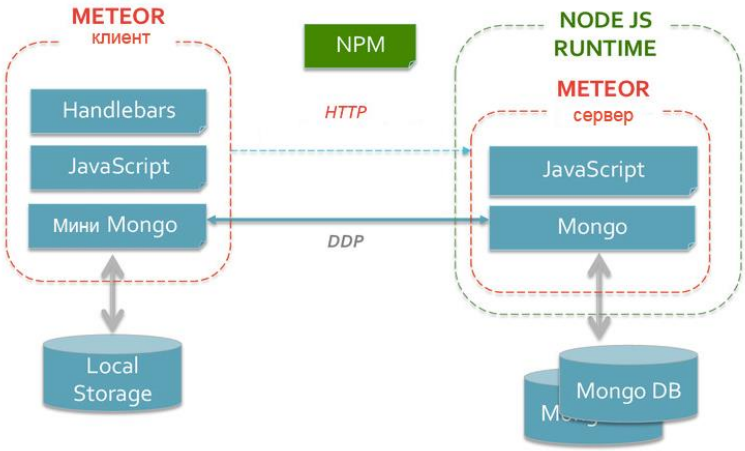
METEOR JAVASCRIPT

5. MeteorJS

Meteor буюу *MeteorJS* -н хамгийн анхны хувилбар 2012 онд гарч байсан бөгөөд анх *Skybreak* нэртэйгээр нийтэд танилцуулагджээ. Одоогоор *Meteor 1.6* хувилбар гараад байна.

Meteor нь вэб болон мобайл аппликэйшнийг цэвэр *JavaScript* -ээр бүтээхэд зориулсан бүрэн хэмжээний платформ юм. *Meteor* ашиглан бүтээгдсэн аппликэйшны өгөгдөл *MongoDB* -д хадгалагдана. Сервер талд *Node.js*, хэрэглэгчийн гар утасны хөтөч дээр *JavaScript* ажиллана. *MeteorJS* фрэймворкын тусламжтайгаар өөрийн програмын сервер болон клиент талыг *JavaScript* -ээр бүрэн бичиж болно [2].

Meteor нь гар утасны санах ойд өөрийн *мини Mongo* өгөгдлийн санг ашиглан өгөгдлийг хуулбарлан ажиллуулдаг. Гар утаснаас серверт хүрэх бүх холболт болон синхрончлолыг *Meteor* хангана. Харин *мини Mongo* нь *MongoDB API* -н JavaScript хэрэгжүүлэлт юм.



Зураг 22. Meteor ерөнхий бүтэц

Meteor автомат синхрончлолоор бүрэн хангах учир тухайн програмын хөгжүүлэлтийг тусдаа буюу клиент ба сервер гэж салгах шаардлагагүй болно. Өөрөөр хэлбэл клиент талын код,

сервер талын код нэг ижил *JavaScript* файлд байрлана.

Meteor нь *Apache Cordova* -г ашиглан програмыг хөрвүүлдэг учир *IOS* болон *Android* -д зориулж тусгай код бичих шаардлагагүй. Энэ нь вэб болон эх кодын холимогийг ажиллуулдаг тул өндөр чанартай аппликэйшныг хүргэдэг.

MeteorJS давуу болон дутагдалтай талыг доор дурдая.

- *Meteor* нь хурдан хөгжүүлэх боломж олгодог;
- Зөвхөн нэг хэл дээр хөгжүүлэх боломжтой;
- Сурахад хялбар;
- *Meteor нь real-time* програм хангамж бүтээхийг хүсэгчдэд зориулсан төгс шийдэл;
- Өгөгдлийн сангаас бүх аппликэйшний бүх загварыг автоматаар шинэчилдэг.
- *Meteor* бүрэн хэмжээний платформ учир *back-end* код болон *front-end* кодыг өөрөө сонгон бичих боломжтой. Үүнийг сонгохын тулд *ReactJS* болон *AngularJS* хамтарч ажиллна.

5.1. Үйлдлийн системийн шаардлага

MeteorJS платформыг ашиглахын тулд хэрэглэгч таны компьютер дараах үйлдлийн системийн шаардлагыг хангасан байх ёстой. Үүнд:

- *Linux*: CentOS 6.4 болон хойших хувилбар
- *Windows*: Windows 7/Windows Server 2008 R2 болон хойших хувилбар

5.2. Суулгах програмчлалын хэрэгслүүд

Meteor -г суулгахаас өмнө хэд хэдэн хэрэгслийг суулгах шаардлагатай. Үүнд: *Node.js* болон *NPM* програмууд багтана. Мөн *Meteor нь* өгөгдлийг *MongoDB* өгөгдлийн санд хадгалдаг. *MongoDB* -н тухай номны эхний хэсэгт тайлбарласан болно.

5.2.1 NPM суулгах

Node.js нь *NPM* -тэй цуг ирдэг бөгөөд хэрхэн суулгах талаар энэ номны эхний хэсэгт тайлбарласан болно. Үүнээс гадна *Meteor* -н *NPM* -г ашиглаж болно. Үүнийг суулгахын тулд дараах командыг *Command Prompt* дээр бичнэ.

```
Meteor add meteorhacks:npm
```

5.2.2 Meteor суулгах

Windows үйлдлийн систем дээр *Chocolatey* -г суулгана. Үүний дараа дараах командыг *Command Prompt* дээр бичиж ажиллуулна.

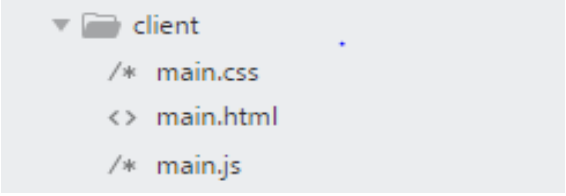
```
Choco install meteor
```

Харин *iOS* эсвэл *Linux* үйлдлийн систем дээр өөрийн терминалаас хамгийн сүүлийн албан ёсны *Meteor* хувилбарыг суулгана.

```
Curl https://install.meteor.com/ | sh
```

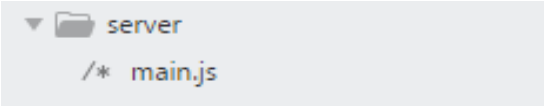
5.2.3 Клиент болон сервер дээр

Meteor нь сангуудаа импортлож оруулж ирдэг. Түүнээс гадна бүх файлыг импортоор оруулж ирэх боломжтой. *Meteor* нь зөвхөн *JavaScript* импорт хийх боломжийг олгодог. Шинэ үүсгэсэн төслийн клиент хавтас дараах зурагт харуулсан бүтэцтэй байна.



Зураг 23. MeteorJS клиент хавтас

Харин шинэ үүсгэсэн төслийн сервер хавтас дараах байдалтай харагдана.



Зураг 24. MeteorJS сервер хавтас

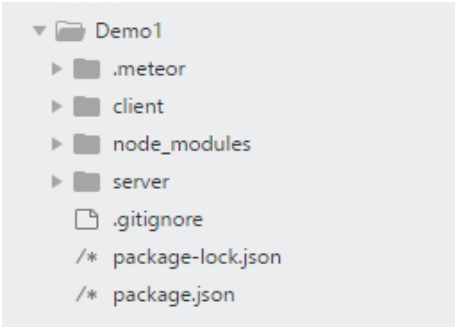
5.3. Ашиглагдах функц үйлдлүүд: MeteorJS жишээ 1

5.3.1 Шинэ аппликэйшн үүсгэх

Аппликэйшн үүсгэхийн тулд *Command Prompt* -г нээгээд дараах командыг бичнэ.

```
meteor create Demo1
```

Командын биелэлт нь дараах байдлаар үүснэ. *Meteor* аппликэйшн нь шаардлагатай бүх файлыг үүсгэнэ.



Зураг 25. Үүссэн Meteor апп

Үүссэн төслөө дараах командаар дуудаж ажиллуулна.

```
cd Demo1
meteor
```

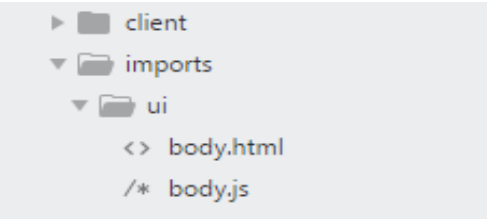
5.3.2 Загвар

Эхлээд үүссэн клиент дотор **main.html** хуудаснаас **<head>** тагыг үлдээж **<body>** тагыг устгана.

```
<head > //апп-н толгой хэсэг
  <title>Demo1</title>
</head>
```

Зураг 26. MeteorJS дэх main.html хуудасны жишээ

Одоо бид шинээр импорт хавтас үүсгээд дотор нь **ui** буюу хэрэглэгчийн интерфейс хавтасыг үүсгэнэ. Хэрэглэгчийн интерфейс **хавтсанд body.html, body.js** хуудсыг үүсгэнэ.



Зураг 27. MeteorJS хавтсанд шинэ хавтас нэмэх

body.html хуудсанд дараах кодыг бичье. Энэ хуудас нь *JavaScript* -д байгаа массивийн өгөгдлийг хөтөч дээр мөр мөрөөр нь харуулах кодын хэсэг юм.

```
<body> <!--body тагын эхлэл апп-н ерөнхий хэсэг-->
  <div class="container"> <!--container нэртэй div
  таг үүсгэж байна-->
    <header><!--толгой хэсэгт харагдах гарчиг
хэсэг-->
      <h1>Demo1</h1>
        <form class="new-task">
          <input type="text" name="text"
placeholder="Type to add new tasks" />
        </form>
      </header>
    <ul>
      {{#each tasks}}<!--tasks дотор өгөгдлийг нэг
нэгээр авах-->
        {{> task}} <!-- template -ийн өгөгдлийг
авч байна-->
  {{/each}}
   </ul>
```

```
  </div>
</body>
<!--body.js хуудастай холбогдож tasks массивийн
text -д байгаа өгөгдлийг авч байна.-->
<template name="task"><li>{{text}}</li></template>
```

Зураг 28. MeteorJs жишээний main.html файл

body.js хуудсанд дараах кодыг бичье. Энэ нь **template.body** доторх **tasks** нэртэй массивт **text** хувьсагчид утга оноож байна. Энэ утгуд хэрэглэгчид харагдана.

```
//html дээрх template тагыг ашиглахын тулд
импортлох.
import { Template } from 'meteor/templating';
//body.html өгөгдөл дамжуулахын тулд импортлох.
import './body.html';
Template.body.helpers({
    //tasks нэртэй массив үүсгэх text хувьсагчид
утга олгож байна.
  tasks: [
  { text: 'This is task 1'},
  { text: 'This is task 2'},
  { text: 'This is task 3'},
  ]
});
```

Зураг 29. MeteorJS жишээний body.js файл

Meteor нь *HTML* файлуудыг харуулдаг бөгөөд дээд түвшиний 3 тагийг тодорхойлдог. Үүнд: <head> <body> <template>

- **<head >** таг дотор бичсэн бүх зүйл илгээсэн HTML -ийн толгой хэсэг рүү нэмэгдэнэ.
- **<body>** таг дотор бичсэн зүйлс хөтөч дээр харагдах хэрэглэгчийн интерфейс болно. Мөн **body** хэсэг

Template.body -r ашиглан **JavaScript** кодыг лавлаж болно. Үүнийг тусгай "эцэг" загвар гэж үзээд бусад удамшсан загварыг оруулж болно.

- **<template>** таг нь HTML доторх **{{> templateName}}** болон **template.template** -г ашиглан өөрийн скрипттэй холбогдоно.

Одоо харин үүсгэсэн хуудсаа **client/main.js** дуудаж өгнө.

```
import '../imports/ui/body.js';
```

Загварыг илүү сайн болгохын тулд **css** загваруудыг тодорхойлж өгье. Дараах код бүх *HTML* хуудасны тагуудад хамааралтай.

```
body {
  font-family: sans-serif;
  background-color: #315481;
  background-image: linear-gradient(to bottom,
#315481, #918e82 100%);
  background-attachment: fixed;
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;

  padding: 0;
  margin: 0;
  font-size: 14px;
}
.container {
  max-width: 600px;
  margin: 0 auto;
  min-height: 100%;
  background: white;
}
```

```
header {
  background: #d2edf4;
  background-image: linear-gradient(to bottom,
#d0edf5, #e1e5f0 100%);
  padding: 20px 15px 15px 15px;
  position: relative;
}
#login-buttons {
  display: block;
}
h1 {
  font-size: 1.5em;
  margin: 0;
  margin-bottom: 10px;
  display: inline-block;
  margin-right: 1em;
}
form {
  margin-top: 10px;
  margin-bottom: -10px;
  position: relative;
}
.new-task input {
  box-sizing: border-box;
  padding: 10px 0;
  background: transparent;
  border: none;
  width: 100%;
  padding-right: 80px;
  font-size: 1em;
}
.new-task input:focus{
  outline: 0;
}
```

```
ul {
  margin: 0;
  padding: 0;
  background: white;
}
.delete {
  float: right;
  font-weight: bold;
  background: none;
  font-size: 1em;
  border: none;
  position: relative;
}
li {
  position: relative;
  list-style: none;
  padding: 15px;
  border-bottom: #eee solid 1px;
}
li .text {
  margin-left: 10px;
}
li.checked {
  color: #888;
}
li.checked .text {
  text-decoration: line-through;
}
li.private {
  background: #eee;
  border-color: #ddd;
}
header .hide-completed {
  float: right;
```

```
}
.toggle-private {
  margin-left: 5px;
}
@media (max-width: 600px) {
  li {
    padding: 12px 15px;
  }
  .search {
    width: 150px;
    clear: both;
  }
  .new-task input {
    padding-bottom: 5px;
  }
}
```

Зураг 30. MeteorJs жишээнд ашиглаж буй ***css*** *файл*

5.3.3 Цуглуулга үүсгэх

Цуглуулга нь *Meteor* -н байнгын өгөгдлийг хадгална. *Meteor* дахь цуглуулгын гол онцлог нь сервер болон клиентээс хандаж болдог. Ингэснээр олон сервер код бичилгүйгээр логикийг бичихэд хялбар байдаг. Мөн цуглуулга дээрх хамгийн сүүлийн үеийн мэдээллийг автоматаар харуулдаг.

```
MyCollection = new Mongo.Collection("My-
collection");
```

Шинэ цуглуулга үүсгэх нь хялбар бөгөөд сервер дээр *MongoDB* цуглуулга үүсэх ба клиент талд серверийн цуглуулгат холбогдсон кешийг үүсгэдэг. Цуглуулга үүсгэхийн тулд **imports** хавтас дотор **api** хавтас үүсгээд дотор нь **tasks.js** нэртэй хуудсыг үүсгэе.

```
//MongoDB өгөгдлийн сангийн санг импорлож байна
import {Mongo} from 'meteor/mongo';
//tasks нэртэй цуглуулга үүсгэх
export const Tasks = new
Mongo.Collection('tasks');
```

Зураг 31. MeteorJS жишээний tasks.js файл

Энэ модулийг сервер дээр импортлож оруулна. **Server/main.js**

```
// tasks.js импортлох
 import '../imports/api/tasks.js';
```

Зураг 32. MeteorJS жишээнд task.js файлыг импортлох

Цуглуулгат өгөгдлийг командаар бичиж оруулахын тулд клиент талын *JavaScript* кодыг шинэчилнэ.

```
// tasks.js импортлох
 import { Tasks } from '../api/tasks.js';
 Template.body.helpers({
     // өгөгдлийг дэлгэцэнд харуулах хэсэг
   tasks() {
    return Tasks.find({});
  }
});
```

Зураг 33. MeteorJS жишээний body.js файлыг шинэчлэх

Цуглуулга доторх зүйлсийг *баримт* гэж нэрлэдэг. *Command Prompt* нээж өөрийн апп -н санд хүрэхийн тулд дараах командыг ажиллуулна.

```
C:\Users\u\Desktop\Demo1>meteor mongo
```

Tasks -н *text* -д “Сайн байна уу, Эх дэлхий минь.” гэсэн утга оноох болон системийн одоогийн цагийг авч буй кодын жишээд дор харуулав.

```
db.tasks.insert({ text:”Сайн байна уу, Эх дэлхий
минь.” , createdAt: new Date() });
```

Зураг 34. MeteorJS жишээний tasks.js файл

5.3.4 Форм болон эвентүүд нэмэх

```
<!-- Шинэ оролтын талбар үүсгэж байна -->
<form class="new-task">
   <input type="text" name="text" placeholder="Энд
шинэ тэмдэглэл бичнэ үү" />
 </form>
```

Зураг 35. MeteorJS жишээний body.html файлын өөрчлөлт

Үүнийг ашиглах эвентийг сонгохын тулд *JavaScript* кодыг нэмэж өгнө.

```
//шинэ үйлдэл нэмэх
Template.body.events({
  'submit .new-task'(event) {
    event.preventDefault();
    const target = event.target;
    // оролтын утгыг авах
    const text = target.text.value;
    // Цуглуулгыг оруулах
    Tasks.insert({
      text,
      // одоогийн цагыг авах
      createdAt: new Date(),
    });
    target.text.value = '';
  }
})
```

Зураг 36. MeteorJS жишээний body.js файлыг шинэчлэх

createdAt талбарыг ашиглан үр дүнг харуулна. Үүний тулд дараах даалгаврыг нэмнэ.

```
tasks() { // Дэлгэцэнд оруулсан өгөгдлийг харуулах
 return Tasks.find({}, {sort: {createdAt: -1 } });
}
```

Зураг 37. MeteorJS жишээ файлын tasks функц

5.3.5 Шалгах болон устгах үйлдэл нэмэх

Imports -н *ui* хавтас дотор *task.html* хуудас үүсгэн дараах кодыг бичнэ. Энэ хэрэглэгчийн оруулсан утгыг шалгах болон устгахад ашиглана.

```
<!--task нэртэй темплайт үүсгэж байна-->
<template name="task">
    <!--шалгах-->
  <li class="{{#if checked}}checked{{/if}}">
    <!—устгах товч -->
    <button class="delete">&times;</button>
    <!--шалгах товч-->
    <input type="checkbox" checked="{{checked}}"
class="toggle-checked"/>
    <span class="text">{{text}}</span>
  </li>
</template>
```

Зураг 38. MeteorJS жишээ файлын tasks.html

Imports доторх **ui** хавтсанд буй **task.js** -д дараах кодыг бичье. Өмнөх бичсэн үйлдлүүдийг устгана. Мөн **body.html** хуудас доторх **<template name=”task”>..</template>** хэсгийг устгана.

```
//темплэйтийг импортлох
import {Template} from 'meteor/templating';
//task.js -г импортлох
import {Tasks} from '../api/tasks.js';
 //task.html -г импортлох
import './task.html';
```

```
Template.task.events({
'click .toggle-checked'() {
        $set: { checked:  !this.checke },
  },
  'click .delete'() {
    Tasks.remove(this._id);
  },
});
```

Зураг 39. MeteorJS жишээ файлын task.js

Одоо **body.js** хуудсанд **task.js** импортлоно.

```
//task.js -г импортлох
import './task.js';
```

Зураг 40. MeteorJS жишээ body.js файлд task.js файлыг импортлох

5.3.6 Хэрэглэгчийн интерфейсийн төлөвийг reactive dictionary -д хадгалах

Үүнд хэрэглэгч шүүлт хийх боломжийг олгоно. Ингэснээр хэрэглэгч зөвхөн бүрэн бус ажиллагааг харах боломжтой болно.

```
<label class="hide-completed"><!—-Шинэ шалгах товч
үүсгэж байна-->
    <input type="checkbox" />
    Hide
</label>
```

Зураг 41. MeteorJS жишээний хэрэглэгчийн интерфейсийн төлөв

Одоо ReactJS багцыг нэмнэ.

```
C:\Users\u\Desktop\Demo1>meteor add
reactive_dict
```

```
import {ReactiveDict} from 'meteor/reactive-dict';
```

Зураг 42. MeteorJS -д ReactJSив багцыг нэмэх

Сонголтыг шалгах үйлдлийг бичихийн тулд *ReactiveDict* хувьсагчийг шалгах хэрэгтэй. Үүнд *onCreated* функцийг ашиглав.

```
Template.body.onCreated(function bodyOnCreated() {
  This.state = new ReactiveDict();
});
```

Зураг 43. MeteorJS жишээний body.js файлын өөрчлөлт

Одоо харин *template.body.helpers* -г шинэчлэнэ. Доорх код нь *checkbox* шалгагдсан тохиолдолд *task* -г шүүх жишээ юм. Body.js

```
Target.text.value = ‘’,
},
‘change .hide-completed input’(event, instance) {
  Instance.state.set(‘hideCompleted’,
event.targed.checked);
},
```

Зураг 44. MeteorJS жишээний checkbox шалгах хэсэг

Хэрвээ *checkbox* -г шалгавал *tasks* жагсаалтыг харуулна.

```
tasks() {
const instance = Template.instance();
if (instance.state.get('hideCompleted')) {
    // Хэрэв нуусан зүйл шалгагдсан бол tasks -г
шүүнэ

    return Tasks.find({checked: { $ne: true }}, {
sort: { createdAt: -1 } });
}
```

Зураг 45. MeteorJS жишээний tasks функц

Шалгагдаагүй даалгаврын тоог харуулахтай адил хүсэлтийг

ашиглана. Үүнийг дараах жишээгээр харуулья.

```
return Tasks.find({}, {sort: { createdAt: -1} });
},
incompleteCount() {
return Tasks.find({checked: {$ne: true }
}).count();
},
```

Зураг 46. MeteorJS жишээний body.js файл дахь шалгагдаагүй өгөгдлийг тоолох кодын хэсэг

Тоолох функцийг гарчигны хажууд харуулах кодын хэсгийг энд харуулж байна.

```
<h1>Demo1 ({{incompleteCount}})</h1>
```

Зураг 47. MeteorJS жишээний body.html файл дахь өөрчлөлт

5.3.7 Хэрэглэгчийн бүртгэл нэмэх

Meteor хэрэглэгчийн бүртгэлийн хэсгийг хэдхэн минутанд гаргах боломжтой хэрэглэгчийн интерфэйстэй. Үүнийг идэвхжүүлэхийн тулд дараах багцыг нэмж, хэрэглэгчийн бүртгэл үүсгэх командыг бичнэ.

```
C:\Users\u\Desktop\Demo1>meteor add accounts-ui accounts-password
```

Нэвтрэх кодыг оруулахын тулд дараах кодыг *body.html* хуудсанд *chechbox* дор нэмж өгнө.

```
{{> loginButtons}}
```

Meteor -н бүртгэлийн хуудсанд и-мейл хаягийг авдаг тул и-мейл хаягийн оронд хэрэглэгчийн нэрийг ашиглах тохиргоог хийх хэрэгтэй. Үүний тулд *imports* хавтсанд *startup* нэртэй хавтас үүсгэж түүн дотроо *accounts-config.js* хуудсыг үүсгэж дараах кодыг бичнэ.

```
//хэрэглэгчийн бүртгэлийг импортлох
import { Accounts } from 'meteor/accounts-base';
Accounts.ui.config({
//хэрэглэгчийн бүртгэлд хэрэглэгчийн нэрийг авах
  passwordSignupFields: 'USERNAME_ONLY',
});
```

Зураг 48. Хэрэглэгчийн бүртгэлд нэрийг нэмэх кодын хэсэг

Одоо үүнийг *main.js* дээр дуудна.

```
Import ‘../imports/startup/accounts-config.js’;
Import ‘../import/ui/body.js’;
```

Зураг 49. accounts-config.js -г main.js –д импортлох кодын хэсэг

Одоо хэрэглэгчид бүртгэл үүсгээд аппликэйшн руу нэвтэрч болно. Хэрэглэгч нэвтэрсэн тохиолдолд шинэ талбар харуулах, бүртгэл болгонд ямар хэрэглэгч үүсгэснийг харуулах кодыг бичнэ. Үүнийг хийхийн тулд цуглуулгын 2 талбарыг нэмж оруулна.

```
//хэрэглэгчийн бүртгэлийг импортлох
import { Accounts } from 'meteor/accounts-base';
Tasks.insert({
    text,
    //одоогийн цагийг авч байна
    createdAt: new Date(),
    //хэрэглэгчийн id авах
    owner: Meteor.userId(),

    //хэрэглэгчийн нэрийг авах
    username: Meteor.user().username,
});
},
```

Зураг 50. Хэрэглэгчийн нэрийг өгөгдлийн санд хадгалах кодын хэсэг

Хэрэглэгч нэвтэрсэн тохиолдолд **<form>** таг хэрэглэгчид харагдана.

```
<!—-Хэрэглэгч нэвтэрсэн тохиолдолд харагдах цонх--
>
{{#if currentUser}}
  <form class="new-task">
   <input type="text" name="text" place-
holder="Type to add new tasks" />
  </form>
{{/if}}
```

*Зураг 51. Хэрэглэгч нэвтэрсэн тохиолдолд харагдах **<form>** таг*

Одоо хэрэглэгч нэвтэрсэн тохиолдолд хэрэглэгчийн нэрийг харуулах командыг *task.html* хуудсанд бичиж өгнө.

```
<span class=”text”><strong>{{username}}</strong> -
{{text}}</span>
```

Зураг 52. task.html хуудасны өөрчлөлт

5.3.8 Аюулгүй байдал

Өмнө хийсэн зүйлсд хэрэглэгч мэдээллийн сангийн аль ч хэсгийг засах боломжтой байсан. Шинээр үүсгэсэн аппликэйшнд найдвартүй багцыг нэмсэн байдаг. Энэ нь клиентээс өгөгдлийн санг засварлах боломжтой багц юм. Энэ багцыг устгахын тулд дараах командыг ажиллуулна. Хэрэв энэ багцыг устгасны дараа аппликэйшнийг ашиглавал ороолт эсвэл товчны аль нь ч ажиллахгүй. Учир бүх хэрэглэгчийн өгөгдлийн сангийн зөвшөөрөл хүчингүй болсон байгаа.

```
C:\Users\u\Desktop\Demo1>meteor remove
insecure
```

Одоо дараах кодыг *tasks.js* дээр бичнэ. Үүнийг *Meteor* -н арга барилаар бус өөрсдийнхөө арга барилаар тодорхойлсон болно.

```
//MongoDB өгөгдлийн сангийн санг импорлох
import {Mongo} from 'meteor/mongo';
//tasks нэртэй цуглуулга үүсгэх
export const Tasks = new
Mongo.Collection('tasks');
Meteor.methods({
  'tasks.insert'(text) {
    //tasks -н оруулсан утгуудыг шалгах
    check(text, String);
    //хэрэглэгчийн нэвтэрсэн эсэхийг шалгах
    if (! Meteor.userId()) {
      throw new Meteor.Error('not-authorized');
    }
    Tasks.insert({
      text,
      //одоогийн цагийг авах
      createdAt: new Date(),
      //хэрэглэгчийн id авах
      owner: Meteor.userId(),
      //хэрэглэгчийн нэрийг авах
      username: Meteor.user().username,
    });
  },
//өгөгдлийг устгахдаа taskid шалгах
'tasks.remove'(taskId) {
    check(taskId, String);
    Tasks.remove(taskId); //өгөгдлийг устгах
  },
  'tasks.setChecked'(taskId, setChecked)
{//өгөгдлийг шалгах
    check(taskId, String);
    check(setChecked, Boolean);
    //өөрчлөх
    Tasks.update(taskId, { $set: { checked:
```

```
setChecked } });
  },
});
```

Зураг 53. tasks.js файл дахь үйлдлүүдийг өөрийн аргаар тодорхойлох кодын хэсэг

Дараах кодыг *body.js* хуудсанд бичиж өгнө. Энэ нь *tasks.js* тодорхойлсон зүйлийг дуудаж ашиглах боломжийг олгоно.

```
//tasks.insert-г дуудаж өгч ашиглаж байна.
Meteor.call('tasks.insert', text);
```

Зураг 54. body.js файлын өөрчлөлт

Шалгах болон устгах үйлдлийг дуудаж ашиглах үйлдлийг дараах кодонд нэмнэ.

```
Template.task.events({
'click .toggle-checked'() {
//шалгахын тулд tasks.setChecked -г дуудаж ашиглаж
байна.
   Meteor.call('tasks.setChecked', this._id,
!this.checked);
},
'click .delete'() {
//устгахын тулд tasks-remove функцыг дуудаж
ашиглаж байна
Meteor.call('tasks.remove', this._id);
},
});
```

Зураг 55. tasks.js файлын өөрчлөлт

5.4. MeteorJS жишээ 2

Өмнөх жишээн *Meteor* -г ашиглан хийсэн бол энэ демо програм дээр *AngularJS* -г ашигласан програм хийнэ. *AngularJS* дээр *front-end* кодыг бичиж хөгжүүлэлт хийдэг.

Жишээндээ зориулж шинээр прожект үүсгэе. Өмнөх жишээнд шинэ прожест үүсгэсэн дарааллыг санаж прожектийг үүсгэнэ. Энэ удаад *Angular_meteor* гэсэн прожектийг үүсгэсэн гэж үзээд дараагийн алхмуудыг хийе. *AngularJS* -г ашиглахын тулд эхлээд *Meteor* -н хэрэглэгчийн интерфейс багцыг устгах хэрэгтэй. Энэ багцыг *blaze* гэж нэрлэдэг.

```
C:\Users\u\Desktop\Angular_meteor>meteor
remove blaze-html-templates
```

Одоо *AngularJS* ашиглахад хэрэгтэй пакетыг *Meteor* -н багцад нэмнэ.

```
C:\Users\u\Desktop\Angular_meteor>meteor    add
angular-templates
```

AngularJS -г *Meteor*-той хамтарч ашиглахын тулд зарим *NPM* багцыг нэмж өгнө.

```
C:\Users\u\Desktop\Angular_meteor>meteor    npm
install –save angular angula-meteor
```

AngularJS - *Meteor* багц *HTML* файлуудыг бүхэлд нь задлан тэдгээрийн бүх замыг *AngularJS* загвар кешээр сольдог.

```
<head>  <!-- апп-н толгой хэсэг -->
  <title>Angular_meteor</title>
</head>
<body> <!-- апп-н их бие хэсэг буюу дотор нь

загваруудыг бичиж өгнө -->
  <div class="container" ng-app="simple-todos">
  </div>
</body>
```

Зураг 56. Angular_meteor жишээний HTML хуудас

```
//Angular ашиглахын тулд импортлож байна.
import angular from 'angular';
//Angular-meteor -г импортлож байна.
import angularMeteor from 'angular-meteor';
//simpletodos болон массивт Angular модуль үүсгэж
байна
angular.module('simple-todos', [
  angularMeteor,
]);
```

Зураг 57. main.js файлд AngularJS санг импортлох кодын хэсэг

Одоо импорт нэртэй сан үүсгэх хэрэгтэй энэ нь бусад сангуудаас өөрөөр ажилна. Импорт сангаас гадна файлууд нь *Meteor* серверийн эхлэх үед автоматаар дуудагдах болно. Импорт сан доторх файлууд зөвхөн импорт мөрийг ачаалахад дуудагддаг. Импорт үүсгэсний дараа дахин хоёр шинэ сан үүсгэх болно.

Imports/compontents/todosList/todosList.html нэртэй хуудас үүсгэж дараах кодыг бичнэ. Энэ нь **todosList.js** дотор байрлах **tasks** массивт байгаа өгөгдлийг авч жагсаалт маягаар харуулна.

```
<header><!--толгой хэсгийн гарчигийг агуулж байна-
->
   <!-- гарчиг хэсэг-->
   <h1>Todo List</h1>
</header>

<ul>
  <li ng-repeat="task in $ctrl.tasks">
    {{task.text}}
  </li>
</ul>
```

Зураг 58. ***todosList.html*** *файл*

```
//AngularJS -н санг импортлож байна
import angular from 'angular';
//AngularJS-Meteor санг импортлож байна
import angularMeteor from 'angular-meteor';
import template from './todosList.html';
class TodosListCtrl {
  constructor() {
  //contructor нэртэй классд task массив үүсгэж
байна
  this.tasks = [
  //үүсгэсэн массивт text төрөлд утга оноож байна
  { text: 'This is task 1'},
  { text: 'This is task 2'},
  { text: 'This is task 3'},
  ];
 }
}

export default angular.module('todosList', [
    angularMeteor
  ])
    .component('todosList', {
      templateUrl: 'imports/compo-
nents/todosList/todosList.html',
      controller: TodosListCtrl
  });
```

Зураг 59. todosList.js файлын массивт өгөгдөл оруулах хэсэг

Одоо үүнийг хэрэглэгчид харуулахын тулд **`main.html`** файлд дараах кодыг өөрчилнө.

```
<body>
  <div class="container" ng-app="simple-todos">
    <todos-list></todos-list>
  </div><body>
```

```
  <div class="container" ng-app="simple-todos">
    <todos-list></todos-list>
  </div>
</body>
```

Зураг 60. main.html файлын өөрчлөлт

Дараа нь програмд модуль нэмнэ.

```
//simpletodos болон массивт AngularJS модуль
үүсгэж байна
angular.module('simple-todos', [
  angularMeteor,
  todosList.name,
]);
```

Зураг 61. AngularJS модулийг нэмэх

Харин энэ програмд өмнөх жишээний **css** загварыг ашиглана.

5.4.2 Цуглуулга дотор tasks -г хадгалах

Өмнөх жишээнд хийсэнтэй адил шинэ цуглуулга үүсгэхдээ *imports* сан дотор *api* нэртэй санг үүсгэнэ. Одоо *tasks.js* хуудсыг үүсгэж өгөгдлийн санг өмнөхтэй адил импортлож үүсгэнэ. Мөн үүсгэсэн хуудсаа сервер *main.js* хуудсанд импортлож оруулна. Үүний дараа өгөгдлийг өөрөө оруулж өгөхийн тулд *todosList.js* хуудсны кодыг бага зэрэг өөрчлөх шаардлагатай.

```
class TodosListCtrl {
constructor($scope) {
$scope.viewModel(this);
this.helpers({
    tasks() { //өгөгдөл дэлгэцэнд харуулах хэсэг
    return Tasks.find({});    }
      })
}
```

```
export default angular.module('todosList', [
  angularMeteor
])
  .component('todosList', {
    templateUrl:
'imports/components/todosList/todosList.html',
    controller: TodosListCtrl
});
```

Зураг 62. todosList.js -н controller-ийн кодыг өөрчлөх хэсэг

Одоо *Command Prompt* -с *tasks* өгөгдөл нэмэж үзье. Ингэхийн тулд дараах командыг бичнэ. Meteor -н өгөгдлийн санг дуудаж ажиллуулах команд:

```
C:\Users\u\Desktop\Angular_meteor>meteor mongo
```

Command Prompt -с цуглуулгад өгөгдөл оруулах команд:

```
Db.tasks.insert({ text:”hello world”,
createdAt: new Date() });
```

5.4.3 Форм болон эвентүүд нэмэх

Энэ хэсэг хэрэглэгчийн үйлдлийг өргөтгөх формуудыг нэмнэ.

```
<form class="new-task" ng-
submit="$ctrl.addTask($ctrl.newTask)">
<input ng-model="$ctrl.newTask" type="text"
name="text" placeholder="Type to add new tasks"/>
</form>
```

Зураг 63. todosList.html хуудасны өөрчлөлт

Нэмсэн форм дээр хэрэглэгч утга оруулахад оруулсан утгыг цуглуулгад нэмнэ. Хэрэглэгчийн оруулсан утгыг өгөгдлийн санд хадгалах болон хэрэглэгчийн оруулсан утгыг харуулах кодын хэсгийг доор харуулав.

```
//хэрэглэгчийн оруулсан утгыг цуглуулагат нэмнэ
addTask(newTask) {
    Tasks.insert({
        text: newTask,// хэрэглэгчийн оруулсан
утга
        createdAt: new Date// одоогийн цаг
    });
    this.newTask = '';//цуглуулгад нэмсний дараа
newTask
    }

todosList.js
this.helpers({
     tasks() { //хэрэглэгчийн оруулсан утгыг
харуулах хэсэг
       return Tasks.find({}, {
         sort: {
           createdAt: -1
       }
   });
```

Зураг 64. todosList.js файлын өөрчлөлт

5.4.4 Шалгах болон устгах үйлдлийг нэмэх

Шалгах болон устгах үйлдэл хийхэд устгах болон шалгах товчийг үүсгэж өгнө. Устгах товчийг дарснаар цуглуулга доторх сонгогдсон утга устгагдана.

```
<ul>
  <li ng-repeat="task in $ctrl.tasks" ng-
class="{'checked': task.checked}">
    <button class="delete" ng-
click="$ctrl.removeTask(task)">times;</button>
     <input type="checkbox" ng-
checked="task.checked" ng-
click="$ctrl.setChecked(task)" class="toggle-
```

```
checked"/>
     <span class="text">
      <!-- {{task.text}} -->
      <strong>{{task.username}}</strong> -
{{task.text}}
    </span>
   </li>
</ul>
```

Зураг 65. todosList.html файлын шалгах, устгах үйлдэл хийх товч нэмэх кодын хэсэг

```
setChecked(task) {
    // check -лэгдсэн утгыг одоо байгаа утгын
эсрэгээр тохируулна
    Tasks.update(task._id, {
      $set: { checked: !task.checked },
    });
  }
   removeTask(task) { // устгах функц
    Tasks.remove(task._id);
  }
}
```

Зураг 66. todosList.js html хуудсанд бичигдсэн шалгах, устгах товчны үйлдлүүд

5.4.5 Цуглуулгыг шүүх, шалгах

Энэ хэсэгт хэрэглэгчид харагдах талбарын өгөгдлийг шүүх *checkbox* нэмнэ. *todosList.html* файлд дараах кодыг нэмнэ.

```
<label class = "hide-completed">
    <input type = "chechbox" ng-model =
"$ctrl.hideCompleted"/> Hide completed tasks
</label>
```

Зураг 67. Шалгах товч нэмэх кодын хэсэг

```
//хэрэв дарагдсан бол шалгагдсан
if (this.getReactively('hideCompleted')) {
    selector.checked = {
      $ne: true
    };
  }
```

Зураг 68. todosList.js файлын шалгах товчны арын кодын хэсэг

Бүрэн шалгагдаагүй утгуудын тоог харуулах функцийг бичье.

```
incompleteCount() {
return Tasks.find({
    checked: {//шалгагдсан бол true
        $ne: true
        }
    }).count(); //шалгагдаагүй утгыг тоолох функц
    },
```

Тоолох функцийг гарчигний хажууд дуудаж ажиллуулах код:

```
<!-- шалгагдаагүй утгыг тоолох функцийг дуудаж
байна -->
<h1>Todo List({{$ctrl.incompleteCount}})</h1>
```

5.4.6 Хэрэглэгчийн бүртгэл нэмэх

Энэ хэсэгт хэрэглэгчид зориулсан нэвтрэх хэсгийг хийнэ. *Meteor* нь бүртгэлийн хэсэгтэй байдгийг дээр дурдсан. Бүртгэлийн хэсэг болон хэрэглэгчийн интерфейсийг идэвхжүүлэхийн тулд холбогдох багцуудыг нэмэх болно.

```
C:\Users\u\Desktop\Angular_meteor>meteor add
accounts-password dotansimha:accounts-ui-
angular
```

Accounts-password нь нууц үг дээр тулгуурласан нэвтрэлт танилтын бүх логикийг агуулсан багц юм. **dotansimha:**

accounts-ui-angular хэрэглэгчийн баталгаажуулалтын маягтад шаардлагатай бүх *HTML* болон *CSS* агуулж байгаа **<login-товч>** тушаалыг агуулна.

Одоо модулийн тодорхойлолтод **account.ui** модулийн хамаарлыг нэмнэ хэрэгтэй.

```
Angular.module('simple-todos', [
  angularMeteor,
  todosList.name,
  'accounts.ui'
]);
Function onReady() {}
```

Зураг 69. main.js файлын өөрчлөлт

Нэвтрэх кодыг харуулахын тулд дараах **todosList.html** хуудсын **checkbox** -н дор нэмж өгнө.

```
<login-buttons></login-buttons>
```

Meteor -н бүртгэлийн хуудсанд и-мейл хаягыг авдаг тул и-мейл хаягын оронд хэрэглэгчийн нэрийг ашиглахын тулд *imports* хавтсанд *startup* нэртэй хавтас үүсгээд дотор нь *accounts-config.js* хуудсыг үүсгэж дараах кодыг бичнэ.

```
import { Accounts } from 'meteor/accounts-base';
//хэрэглэгчийн бүртгэлийг импортлож байна.
 Accounts.ui.config({
  passwordSignupFields: 'USERNAME_ONLY',
//хэрэглэгчийн бүртгэлд хэрэглэгчийн нэрийг авах
});
```

Main.js хуудсанд *accounts-config* хуудсыг оруулж өгнө.

```
import { Accounts } from 'meteor/accounts-base';
//хэрэглэгчийн бүртгэлийг импортлож байна.
```

Одоо хэрэглэгчид бүртгэл үүсгэж, аппликэйшн руу нэвтэрч

болно. Хэрэглэгч нэвтэрсэн тохиолдолд шинэ талбар харуулах, бүртгэл болгонд ямар хэрэглэгч үүсгэснийг харуулах кодыг бичнэ. Үүнийг хийхийн тулд цуглуулгын 2 талбарыг нэмж оруулна.

```
Tasks.insert({
    text,
    createdAt: new Date(),
//одоогийн цагийг авах
    owner: Meteor.userId(),
//хэрэглэгчийн id авах
    username: Meteor.user().username,
//хэрэглэгчийн нэрийг авах
    });
},
```

Дараа нь хэрэглэгч нэвтэрсэн тохиолдолд **<from>** харагдуулах тохиолдолд дараах кодыг **todosList.html** сольж бичнэ.

```
<form class = "new-task"
      ng-submit="$ctrl.addTask($ctrl.newTask)"
      ng-show="$ctrl.currentUser">
<input ng-model="$ctrl.newTask" type="text"
      name="text" placeholder="Type to add new
      tasks"/>
</form>
```

Зураг 70. Нэвтэсний дараа форм цонхыг харуулах кодын хэсэг

Эцэст нь текстийн өмнө ажиллаж буй бүх үйлдлийг хэрэглэгчийн талбар дээр харуулах кодыг бичнэ.

```
<span class="text">
    <strong>{{task.username}}</strong> -
{{task.text}}
</span>
```

Зураг 71. Нэвтэрсэн тохиолдолд өгөгдлийг харуулах кодын хэсэг

5.5. MeteorJS жишээ 3

Энэ жишээ өмнөх хоёр жишээ програмаас үйлдлийн хувьд ялгаагүй ч кодын хувьд ялгаатай. Жишээ програм 2 дээр *AngularJS* -тэй хослуулж програм бичсэн бол энэ хэсэгт *ReactJS* -тэй хамтарч нэвтрэх, устгах, нөхцөл шалгах, өгөгдөл нэмэх зэрэг үйлдлүүдийг хийх болно. *ReactJS* нь *back-end* буюу арын кодын хөгжүүлэлтэд ашигладаг платформ юм. Үүнийг *Meteor* -той хослуулах болно.

5.5.1 Шинэ апп үүсгэх

Өмнө жишээ програм 1 дээр гүйцэтгэсэн шинэ төсөл үүсгэх дарааллын дагуу аппликэйшнийг үүсгэнэ.

5.5.2 Загвар

ReactJS -г ашиглахын тулд эхлээд *Command Prompt* нээгээд *ReactJS* ашиглах боломжтой *NPM* багцыг багцууд дотор нэмнэ.

```
C:\Users\u\Desktop\React_meteor>meteor npm
install --save react react-dom
```

Шинэ төсөл үүсгэхэд үүссэн **main.html** хуудасны **body** таг хэсгийг дараах кодоор солино.

```
<head> <!--апп толгой хэсэг -->
  <title>React_meteor</title>
</head>
<body> <!-- апп-н их бие хэсэг буюу дотор нь
загваруудыг бичиж өгнө -->
  <div id="render-target"></div>
</body>
```

Зураг 72. MeteorJS жишээ 3 body.html

Импорт сангаас гадна файлууд *Meteor* серверийг эхлэх үед автоматаар дуудагдах болно. Импортын сан доторх файлууд

нь зөвхөн *import statement* -ийг биелэхэд ачаална.

```
//ReactJS -н санг импортлох
import React from 'react';
//Meteor санг импортлох
import { Meteor } from 'meteor/meteor';
//react-dom ашиглахын тулд импортлох
import { render } from 'react-dom';
//app.js оруулах
import App from '../imports/ui/App.js';
Meteor.startup(() => {
    // Meteor -г эхлүүлэх функц
  render(<App />, document.getElementById('render-
target'));
});
```

Зураг 73. MeteorJS -д сан импортлох

Импортын санг үүсгэсний дараа *Imports/ui/App.js* гэх шинэ файл үүсгэнэ. Үүсгэсэн хуудсанд дараах кодыг бичнэ.

```
//ReactJS-н реак,компонент санг импортлох
import React, { Component } from 'react';
//task.js-н task классыг ашиглахын тулд импортлох
import Task from './Task.js';
//app-н бүрэлдэхүүн хэсэг
export default class App extends Component {
getTasks() {
//getTasks нэртэй функц үүсгэж id болон text утга
буцаах
return [
    { _id: 1, text: 'This is task 1' },
    { _id: 2, text: 'This is task 2' },
    { _id: 3, text: 'This is task 3' },
];
}
```

Зураг 74. App.js файлын массивт өгөгдөл оруулах

Мөн дараах кодыг **App.js** бичиж өгнө. **renderTasks()** функц нь **getTasks()** функцийн **id** -г **key** гэсэн түлхүүрт, **task** -г **task** -д авч байна.

```
renderTasks() {// renderTasks нь getTasks-н id
болон утгуудыг авч байна
    return this.getTasks().map((task) => (
      <Task key={task._id} task={task} />
    ));
  }
```

Зураг 75. App.js файлын getTasks() функц

Үүнийг ч бас **App.js** хуудсанд нэмж өгнө. Энэ хэсгийн код нь *HTML* дээр бичигдэх кодыг **render()** функц ашиглан JavaScript дээр бичиж ажиллуулна. Энэ *header* буюу гарчиг болон **renderTasks()** функцийн буцаалтыг авч байна.

```
  render() {
    return (
      <div className="container">
        <header>
          <h1>Todo List</h1>
        </header>
        <ul>
          {this.renderTasks()}
        </ul>
      </div>
    );
  }
}
```

Зураг 76. ***App.js*** *html таг бичих* ***render()*** *функц*

Дараах код нь **App.js** файл доторх **getTasks** функцийн **text** дэх утгуудыг **renderTasks** функцийн тусламжтай жагсаалт маягаар дэлгэцэнд харуулна.

```
import React, { Component } from 'react';
//ReactJS-н реак,компонент санг импортлох
//Task-н бүрэлдэхүүн хэсэг
export default class Task extends Component {
  render() {
    return (
//App.js доторх getTasks-н утгуудыг renderTasks-н
тусламжтай харуулах хэсэг
      <li>{this.props.task.text}</li>
    );
  }
}
```

Зураг 77. Массивт байгаа утгуудыг дэлгэцэнд харуулах кодын хэсэг

Мөн өмнөх демо програм 1 дээр ашигласан **`css`** кодыг энэ програм дээр ч ашиглах тул **`main.css`** дээр өмнөх **`css`** хуулж бичих хэрэгтэй.

5.5.3 Цуглуулга үүсгэх

Meteor дахь цуглуулгын онцлог нь сервер болон клиентээс хандаж болох бөгөөд ингэснээр маш олон сервер код бичихгүйгээр логикийг бичихэд хялбар байдаг. Өмнөх жишээ 1, 2 програм дээр үүсгэж байсантай адилаар *Meteor* -н цуглуулгыг үүсгэнэ.

Эхлээд үүссэн **`import`** хавтсанд дотор **`api`** хавтас үүсгэж дотох нь **`tasks.js`** файлыг үүсгэнэ. Үүсгэсэн файл дотор дараах цуглуулга үүсгэх кодыг оруулж өгнө.

```
import { Mongo } from 'meteor/mongo';
//MongoDB өгөгдлийн сан импортлох
export const Tasks = new
Mongo.Collection('tasks');
//tasks санг үүсгэж байна
```

*Зураг 78. **Tasks** нэртэй цуглуулга үүсгэх кодын хэсэг*

Үүнийг серверийн **main.js** хуудсанд импортлож оруулж ирнэ.

```
//tasks.js хуудсыг импортлох
import '../imports/api/tasks.js';
```

Зураг 79. Main.js файлд tasks.js -г импортлох

ReactJS компонент доторх цуглуулгаас өгөгдлийг ашиглахын тулд *Атмосфер* багцын солилцооны өгөгдлийг ашиглаж болно. Энэ нь *Meteor* -н *ReactJS* өгөгдлийг *React* -н бүрэлдэхүүн шатлалд шаталсан "өгөгдлийн контейнер" үүсгэх боломжийг олгодог.

```
C:\Users\u\Desktop\Angular_meteor>meteor add
react-meteor-data
```

ReactJS нь *Meteor* өгөгдлийг ашиглахын тулд **withTracker** комбинацыг ашиглана. Ашиглах сангуудыг импортлож байна.

```
import { withTracker } from 'meteor/react-meteor-
data';
import { Tasks } from '../api/tasks.js';
```

*Зураг 80. App.js файлд **withTracker** импортлох*

Өмнө нь **getTasks()** функцэд байх өгөгдлийг жагсаалтаар харуулсан бол одоо энэ функцийг устгаж цуглуулагат байгаа өгөгдлийг харуулах кодыг бичнэ.

```
//апп-н бүрэлдэхүүн хэсэг
class App extends Component {
renderTasks() {//үүсгэсэн цуглуулгын өгөгдлийг
авах функц
    return this.props.tasks.map((task) => (
    <Task key={task._id} task={task} />
    ));
}
//withTracker()-г ашигласан байдал. App.js

export default withTracker(() => {
return {
    tasks: Tasks.find({}).fetch(),
    };
})(App);
```

Зураг 81. App.js файлд withTracker ашиглах

5.5.4 Форм болон эвент нэмэх

Энэ үед хэрэглэгчид жагсаалтыг даалгаруудыг нэмэх оролтын талбар нэмнэ. Эхлээд форм буюу оролтын цонхыг нэмнэ. Үүнийг **app.js** дээр **<header>** таг дотор нэмж өгнө.

```
<form className="new-task"
onSubmit={this.handleSubmit.bind(this)} >
<input
    type="text"
    ref="textInput"
    placeholder="Type to add new tasks"
/>
</form>
```

Зураг 82. Оролт авах <form> таг

Та форм элемент дэх **handleSubmit** гэх арга нь **onSubmit** шинж чанар байгаа эсэхийг тодорхойлно. *ReactJS* нь хөтөчийн

үйл явдлыг мониторингийн үйл явдлийн хэлбэрээр сонсдог. Оролтын элемент нь энэ элементийг элемент рүү хялбархан оруулах боломжийг бүрдүүлдэг.

```
handleSubmit(event) {
event.preventDefault();
// React ref-р дамжуулан текст талбарыг хайх
const text =

ReactDOM.findDOMNode(this.refs.textInput).value.tr
im();
Tasks.insert({
    text,
    createdAt: new Date(), // одоогийн цаг
});
//text-н утгыг хоосон болгох
ReactDOM.findDOMNode(this.refs.textInput).value =
'';
}
```

*Зураг 83. **App.js** -д нэмэх hadleSubmit() функц*

withTracker() функц дахь **tasks** -г дараах кодоор солино.

```
export default withTracker(() => {
return {
    tasks: Tasks.find({}, { sort: { createdAt: -1
} }).fetch(),
};
})(App);
```

Зураг 84. withTracker() функц

5.5.5 Шалгах болон устгах үйлдлүүд

Энэ удаад шинэчилсэн устгах үйлдлүүдийг хийнэ. Дараах код нь устгах болон шалгах товчны үйлдлүүдийг бичиж өгсөн.

```
import React, { Component } from
'react';//ReactJS-н реак,компонент санг импортлох
//Task-н бүрэлдэхүүн хэсэг
export default class Task extends Component {
  toggleChecked() {
    // шалгагдсан өгөгдлийн одоо байгаа утгыг
эсрэгээр солино.

    Tasks.update(this.props.task._id, {
      $set: { checked: !this.props.task.checked },
    });
  }
  deleteThisTask() {// task дахь өгөгдлийг устгах
    Tasks.remove(this.props.task._id);
  }
```

Зураг 85. Устгах болон шалгах үйлдэл

Render() классд үүсгэсэн товчнууд классын нэрийг өөрчлөх код:

```
render() {
//даалгаваруудыг шалгасны дараа классын нэрийг
өөрчилнө.
//Ингэснээр бид тэдгээрийг CSS дээр сайн
загварчилж болно
const taskClassName = this.props.task.checked ?
'checked' : '';
```

Зураг 86. Tasks.js файлын render() функц

Render класс дотор устгах болон шалгах товчыг үүсгэж байна. Үүгээр өгөгдлийн сан дахь өгөгдлийг устгаж эсвэл шалгах болно.

```
return (
    <li className={taskClassName}>
    <button className="delete"
onClick={this.deleteThisTask.bind(this)}>
        ⊖times;
    </button>
    <input
        type="checkbox"
        readOnly

        checked={!!this.props.task.checked}
        onClick={this.toggleChecked.bind(this)}
    />
    <span
className="text">{this.props.task.text}</span>
    </li>
);}}
```

Зураг 87. Устгах, шалгах товчыг render() классд үүсгэх

5.5.6 Цуглуулгыг шүүх, шалгах

Энэ алхам дээр хэрэглэгчийн талбайн өгөгдлийг шүүх апп нэмнэ. Шинээр шалгах товч нэмэх хэрэгтэй. Үүнийг **App.js** хуудасны **render()** функцийн **<header>** хэсэг нэмж өгнө.

```
constructor(props) {
super(props);
this.state = {
    hideCompleted: false,
//hidecompleted анхны төлөв false утгатай байна
};
}
```

Зураг 88. apps.js файл дээрх өөрчлөлт

Хэрэв шалгах товч дээр дарсан тохиолдолд шалгах товчний төлөвийг эсрэгээр сольно. Өөрөөр хэлбэл, *chechbox* -г дарахад

check -лэгдэнэ гэсэн үг. Дахин дарвал цуцлагдана. Үүнийг өгөгдөл шүүхэд ашиглана.

```
// шалгах товч дээр дарахад төлөвийг эсрэгээр
солино.
toggleHideCompleted() {
this.setState({
    hideCompleted: !this.state.hideCompleted,
});
}
```

Зураг 89. Checkbox сонгогдсон эсэх

5.6. Технологийн хэрэглээ

Meteor -г ашигласан томоохон системүүдийг энд танилцуулья.

- *MAZDA*- машин зардаг худалдааны сайт
- *ROCKET CHAT* – вэб чат платформ
- *OCULUS health* – эрүүл мэндийн тусламж үйлчилгээ үзүүлэх
- *CLASSCRAFT* – Анги танхимд тоглох тоглоом
- *PEDLAR* - Их Британид байрладаг олон хүнтэй гэрээ байгуулсан сайт
- *HELLOMONEY* - Хувь хүний санхүү, хөрөнгө оруулалтын менежментийн багц хэрэгсэл
- *GAMERAVEN* – онлайн тоглоом тоглох олон нийтийн сайт
- *FAVRO* - Бодит цагийн зарчимтай орчин үеийн хамтын ажиллагааны апп

Meteor нь хөгжүүлэхэд хялбар ба зарим програмуудтай хамтарч ашиглахад хялбар бөгөөд *Meteor* -г бүх төрлийн вэб аппликэйшн хийхэд ашиглаж болно. Мөн *Meteor* -г *AngularJS* болон *ReactJS* -тэй хамт ашиглаж болно. Үүнийг ашиглахын тулд өөрийн хийж байгаа програмд тодорхой нэг санг дуудаж ажиллуудлах шаардлагатай. Ингэснээр *ReactJS Meteor*,

AngularJS Meteor зэргийг ашиглах боломж бүрдэнэ.

ANGULAR 4

6. Angular 4

Angular бол *TypeScript* хэлэнд суурилсан нээлттэй эхийн *front-end* вэб аппликэйшн платформ юм. *Angular* -н хамгийн анхны хувилбарыг 2009 онд *Brat Tech* ХХК -ны програм хангамжийн инженер *Miško Hevery AngularJS* нэртэйгээр гаргаж байсан.

Үүний дараагаар *Miško Hevery Google* компанид ажилд орсон ба ажлын хэдэн нөхөдтэйгээ нийлэн *Angular* багийг үүсгэн анхны хувилбараа үргэлжлүүлэн хөгжүүлж *Angular 2* хувилбарыг 2016 оны 9 сард, *Angular 4* хувилбарыг 2017 оны 5 сард гаргасан. *Angular* гэдэг үг нь *HTML* тагуудыг илэрхийлэх **<>** буюу өнцгөн хаалт тэмдэгтээс гаралтай.

Бүх төрлийн үйлдлийн систем дээр бүх төрлийн төхөөрөмжинд зориулагдсан хөгжүүлэлтийг нэг бичилтээр хийх боломжтой. Жишээ нь: Linux, Windows, OS X

AngularJS давуу талууд:

- Интерактив клиент талын кодын томоохон хэмжээний аппликэйшнуудад чиглэсэн.
- Зарим тохиолдолд бага код шаарддаг.
- Нэг хуудастай динамик аппликэйшнуудад зориулсан сайн шийдэлтэй.
- Дэвшилтэт туршилтын онцлогууд
- Хурдан хөгжүүлэлтийн процесс
- *Model-View-Controller* - ын тэнцвэр

AngularJS дутагдалтай талууд:

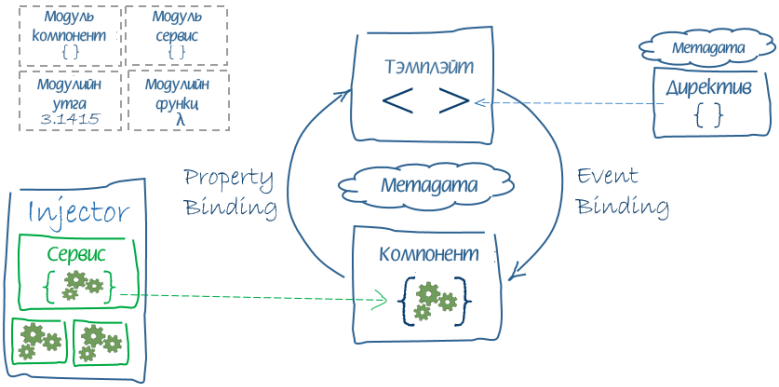
- Асар их өгөгдлийг харуулах үед харьцангуй удаан.

6.1. Бусад ижил төстэй технологитой харьцуулсан судалгаа

	Angular 4	**React**	**VueJS**
Хувилбар	4.0.0	16.2.0	**2.5.9**
Хөгжүүлэгч	Google	Facebook	**Evan You**
Програмчлалын хэл	TypeScript	JSX	**JavaScript/HTML**
Бусад технологиудтай тохиромжтой байх байдал	Сайн	Сайн	**Дунд**
Хэрэгсэл	Сайн	Сайн	**Дунд**
Кодын загвар	JS into HTML	JavaScript Centeric	**JS into HTML**
DOM	Энгийн	Виртуал	**Виртуал**
Пакеж	Дунд	Сайн	**-**
Өгөгдлийн холболт	2 Way	Uni-Directional	**2 Way**
Загвар	TypeScript	JSX	**In HTML**
Архитектур	MVC	View	**MVC**
Rendering	**Сервер тал**	**Сервер тал**	**Хэрэглэгчийн тал**

6.2. Архитектур ба үндсэн ойлголтууд

Angular фреймворкын архитектур нь үндсэн найман хэсгээс тогтох бөгөөд энэ бүлэгт авч үзэх болно. Дараах зургийг харна уу.



Зураг 90. Angular фреймворкын тойм архитектур [3]

6.2.2 Програмчлалын хэлний шаардлага болон суулгах програмчлалын хэрэгслүүд

Angular фреймворк дээр хөгжүүлэлт хийхэд дараах програмчлалын хэлнүүдийг мэдэж байх шаардлагатай. Үүнд: *HTML, CSS, JavaScript, TypeScript. TypeScript* -н тухай *Angular 2* хэсгээс дэлгэрэнгүй харна уу. Харин хэрэгслүүдийн хувьд *Node.js, NPM, Angular CLI, Sublime text* зэргийг ашиглаж болно.

6.2.3 Өгөгдлийн сангийн шаардлага

Angular фреймворк нь *front-end* хөгжүүлэлт хийдэг тул *back-end* фреймворкуудтай холбогдож өгөгдлийн сантай харилцах ба *back-end* фреймворкуудаар дамжуулан бүх төрлийн өгөгдлийн сантай холбогдож ажиллах боломжтой.

- *MongoDB*
- *PostgreSQL*

PostgreSQL нь объект хандалтат өгөгдлийн сан удирдах систем бөгөөд өргөтгөл болон стандартыг дагаж мөрдөхөд анхаарал хандуулдаг юм. Энэ өгөгдлийн сан нь гадаад түлхүүр, холболтууд, харагдацууд, триггерүүдтэй болон

функууд, сторед процедурууд болон бусад өргөтгөлүүдийг дэмждэг. *Unix* болон *Windows* үйлдлийн системүүд дээр ажиллах боломжтой. Үүний давуу талуудыг одоо дурдая.

- Олон интерфэйстэй.
- Цогц өгөгдлийн загваруудтай харьцахад илүү хялбар.
- Түлхүүр утгын хослолыг дэмждэг.
- *JSON* хоёртын сан байгуулах боломжтой.

MySQL нь нээлттэй эхийн өгөгдлийн сан удирдах систем юм. Энэ өгөгдлийн сан нь өндөр шинж чанартай вэбд суурилсан програмуудад илүү тохиромжтой. Мөн энэ нь сервер, клиент болон эмбэдэд системүүд дээр ажилладаг. *MySQL* давуу талууд:

- Хурдтай;
- Сервер, клиент болон эмбэддэд систем дэмждэг;
- Хэрэглэхэд хялбар;
- Хямд;
- Түгээмэл стандарттай;

Apache Cassandra бол ямар ч алдаагүй байх өндөр түвшний боломжоор хангаж, олон төрлийн бүтээгдэхүүний серверүүд дээр их хэмжээний өгөгдөл дамжуулахаар загварчлагдсан үнэгүй, нээлттэй эхийн *NoSQL* өгөгдлийн сан удирдах систем юм. Давуу талууд:

- Өргөтгөх боломжтой;
- Баганан хандалтат;
- Өндөр түвшний гүйцэтгэлтэй;
- Алдаанд тэсвэртэй;

6.2.4 Суурилуулах заавар

Сервер талд *Angular 2* суулгах зааврыг харна уу. Харин клиент талд суулгах боломжгүй.

6.2.5 Програмчлалын хэрэгсэлтэй холбох

Angular 4 нь *front-end* фреймворк бөгөөд *RESTful API* ашиглан бүх төрлийн хэлний *back-end* фреймворктой холбогдож ажиллах боломжтой ба *back-end* фреймворкоор дамжуулан өгөгдлийн сантай холбогдоно. Ашиглаж болох *back-end* фреймворкууд: *Laravel, Symfony, Phalcon, Django, Express, Sails, Ruby and Rails*

RESTful API гэдэг нь вэб аппликэйшнд *HTTP* хүсэлт ашиглан бүх боломжит *CRUD (create, retrieve, update, delete)* үйлдлүүдийг хийх боломжийг олгодог хэрэглээний програм интерфейс юм.

6.3. Ашиглагдах функц үйлдлүүд

6.3.1 Модуль

Angular аппликэйшнууд нь өөрийн гэсэн модульчлагдсан системтэй бөгөөд үүнийг нь *NgModules* гэж нэрлэдэг.

NgModules нь томоохон хэсгээс бүрддэг. Энэ хуудас нь модулиудыг танилцуулах ба *NgModules* хуудас эдгээртэй маш их хамааралтай. *Angular* аппликэйшн бүр хамгийн багадаа нэг *NgModule* класстай бөгөөд үндсэн модулийг нь AppModule гэж нэрлэж хэвшсэн байдаг.

Хэдийгээр *root* модуль нь жижиг програм дахь цорын ганц модуль байж болох ч ихэнх програмууд маш олон онцгой модулиуд, аппликэйшн домайн тус бүрд зориулсан кодын нэгдсэн блок, ажлын урсгал, эсвэл хоорондоо нягт холбоотой багц чадваруудтай байдаг.

NgModule бол *@NgModule decorator* -тай класс юм. *Decorator* бол *JavaScript* классуудыг өөрчлөх функцууд юм. *NgModule* бол модулийн шинж чанарыг тодорхойлох ганц мета өгөгдлийн объект авдаг decorator функц юм. Хамгийн чухал шинж чанарууд нь:

- *declarations* - энэ модульд хамаарах харагдац классууд юм. *Angular* нь 3 төрлийн харагдац классуудтай. Үүнд : компонент, удирдамж болон пайп (pipe).

- *exports* - бусад модулиудын компонентын загварт ашиглагдах болон харагдах боломжтой илэрхийлэх байдлыг тоо хэмжээ
- *imports* - энэ модульд зарлагдсан компонентын загваруудад шаардлагатай бусад модулиудын экспортолсон классууд
- *providers* - энэ модульд глобал сервисүүдийг цуглуулахад туслах сервисийн үүсгэгчид; тэд аппликэйшны бүх хэсэгт хандах боломжтой болсон.
- *bootstrap* - үндсэн аппликэйшны харагдац, *root* компонентыг дуудах, бусад бүх аппликэйшны харагдацыг зохион байгуулах зэрэг багтана. Зөвхөн root модуль *bootstrap* -н шинж чанарыг тохируулах ёстой.

Энгийн *root* модулийн жишээг харна уу.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-
browser';

@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Зураг 91. Энгийн root модулийн жишээ
*(**src/app/app.module.ts**)*

6.3.2 Angular сангууд

Angular нь яг л *JavaScript* модулиудын цуглуулга шиг ба та тэдгээрийг номын сангийн модулиуд гэж төсөөлж болно.

Angular сангийн нэр тус бүр *@angular* гэсэн угтвараар эхэлдэг.

Та тэдгээрийг *NPM* пакет менежер ашиглан суулгах ба импортлон оруулах хэсэг нь *JavaScript* импортын мэдэгдэлтэй байна. Тухайлбал, *@angular/core* сангаас *Angular* -н *Component decorator* -г импортоор оруулах нь дараах байдалтай харагдана.

```
import { Component }  from '@angular/core';
```

Зураг 92. Angular Component импортоор оруулах

6.3.3 Компонент

Компонент нь харагдац гэж нэрлэгдэх дэлгэцийн хэсгийг хянадаг. Жишээлбэл, дараах харагдацуудыг компонент хянадаг.

- Чиглүүлэгч холбоосуудтай **app root**
- Баатруудын жагсаалт
- Баатар засагч

Та класс дотор компонентын аппликэйшн логик болон харагдацыг дэмжихийн тулд юу хийдэг болохыг тодорхойлно. Класс нь шинж чанар болон аргуудын *API* -р дамжуулан харагдацтай нэгтгэдэг.

Тухайлбал, бидний жишээнд гарах *HeroListComponent* нь баатруудын массивыг буцаадаг сервисээс олж авсан баатруудын шинж чанартай. *HeroListComponent* мөн тухайн хэрэглэгч жагсаалтаас баатар сонгон дарах үед сонгогдсон баатрын (*selectedHero*) шинж чанарыг тохируулах *selectedHero()* аргатай.

```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;
constructor(
    private service: HeroService) { }
```

```
  ngOnInit() {
    this.heroes = this.service.getHeroes();  }

  selectHero(hero: Hero) {
    this.selectedHero = hero;}
```

*Зураг 93. HeroListComponent.ts жишээ (**src/app/hero-list.component.ts**)*

Хэрэглэгч аппликэйшныг хэрэглэхэд *Angular* нь компонентуудыг үүсгэж, шинэчилж бас устгадаг. Дээр *ngOnInit()* зарласан шиг таны аппликэйшн амьдралын мужийн утгыг дамжуулан тухайн амьдралын мужийн агшин тус бүрт үйлдэл хийх боломжтой.

6.3.4 Загвар

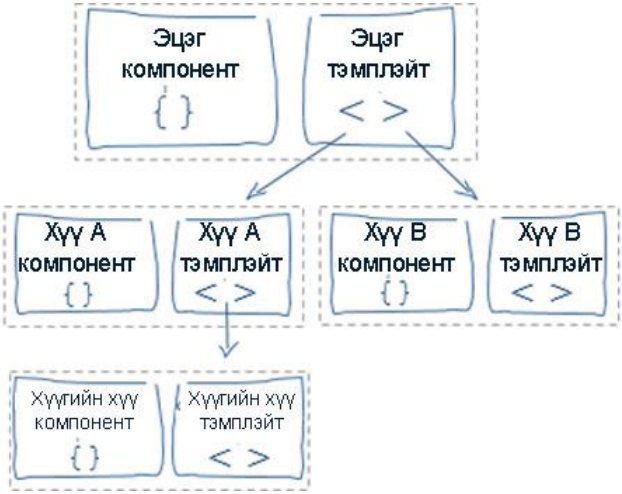
Та компонентын харагдацыг энэ загвараар тодорхойлно. Загвар бол *Angular* компонентыг хэрхэн илэрхийлэхийг хэлдэг *HTML* хэлбэр юм. Загвар нь хэд хэдэн ялгааг эс тооцвол *HTML* шиг харагддаг. *HeroListComponent* -н загварыг доор харуулав.

```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes"
(click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero"
[hero]="selectedHero"></app-hero-detail>
```

*Зураг 94. HeroListComponent.html жишээ (**src/app/hero-list.component.html**)*

Энэ загвар нь *HTML* элементийн **<h2>** болон **<p>** тагуудыг ашиглаж байгаа боловч зарим нэг ялгаа бий. *Angular* загвар нь ***ngFor, {{hero.name}}, (click), [hero], <hero-detail>** гэх мэт синтаксийг хэрэглэдэг. Загварын хамгийн сүүлийн мөрөнд байрлах **<hero-detail>** таг бол *HeroDetailComponent* шинэ компонентыг төлөөлөх тусгай элемент юм.

HeroDetailComponent нь *HeroListComponent* -оос өөр гэдгийг харж болно. *HeroDetailComponent* нь *HeroListComponent* -оор танилцуулсан жагсаалтаас хэрэглэгчийн сонгосон баатрын тухай мэдээллийг дэлгэрүүлнэ. *HeroDetailComponent* бол *HeroListComponent* -н хүү компонент юм. Уламжлалт *HTML* элементүүдийн дунд **<hero-detail>** хэрхэн алдаагүй оршиж байгааг анхааралаарай. Тусгай компонентууд нь ижил *layout* -н уламжлалт *HTML* -тэй ямар нэг хамааралгүй холигддог.



Зураг 95. Root загварт хүү загваруудын тусгай элементүүд [3]

6.3.5 Мета өгөгдөл

Мета өгөгдөл нь *Angular* классыг хэрхэн боловсруулахыг заана. *HeroListComponent* -н кодыг дэлгэрүүлээд харвал энэ

зүгээр л класс болохыг ойлгоно. Тэдгээрт фреймворкын баталгаа байхгүй. Энэ бүгдэд “*Angular*” байхгүй.

Үнэндээ *HeroListComponent* зүгээр л класс юм. Энэ тухай *Angular* -т хэлэх хүртэл энэ компонент биш. Класст мета өгөгдлийг хавсаргаж өгсөнөөр *HeroListComponent* бол компонент гэдгийн *Angular* мэднэ. *TypeScript* дээр *decorator* ашиглан мета өгөгдлийг хавсаргав.

```
@Component(
  selector: 'hero-list',
  templateUrl: 'hero_list_component.html',
  directives: const [CORE_DIRECTIVES,
formDirectives, HeroDetailComponent],
  providers: const [HeroService],
)
class HeroListComponent implements OnInit {
  // ...
}
```

Зураг 96. Decorator ашиглан мета дата өгөгдөл хавсаргах (src/app/hero-list.component.ts (metadata))

@Component decorator нь *Angular* -т компонентыг үүсгэх болон танилцуулахад хэрэгтэй тохиргооны мэдээлэлтэй объектыг шаарддаг. Хамгийн их хэрэглэгддэг *@Component decorator* -н тохиргооны сонголтууд:

- **`selector`**: *CSS selector* нь энэ компонентыг эцэг *HTML* -н хаана **`<hero-list>`** таг байгааг олж үүсгэх болон оруулахыг *Angular* -т хэлнэ.
- **`templateUrl`**: Энэ компонентын *HTML* загварын модулийн харьцангуй хаяг юм.
- **`providers`**: компонентын шаарддаг сервисүүдэд зориулсан хамааралтай оролтын массив. Энэ бол дэлгэцэнд харуулах баатруудын жагсаалтыг авахын тулд компонентын байгуулагч *HeroService* -г шаарддаг

гэдгийг *Angular* -т хэлэх нэг арга юм.

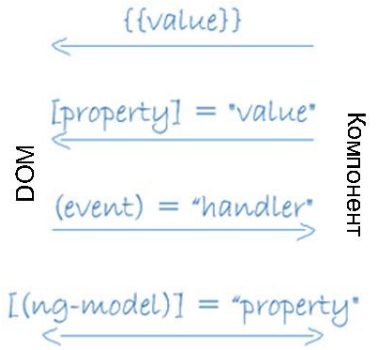
@Component дахь мета өгөгдлүүд нь таны компонентэд зориулсан үндсэн блокуудыг хаанаас авахыг *Angular* -т хэлж өгдөг. Загвар, мета өгөгдөл болон компонент нь хамтдаа харагдацыг тодорхойлдог. Ижил төстэй загварууд дахь бусад мета өгөгдлүүд нь үүнийг хэрэгжүүлсэнээр *Angular* -н зан байдлыг удирдан чиглүүлдэг. *@Injectable*, *@Input* болон *@Output* гэсэн цөөн тооны **`decorator`** -ууд байдаг.

6.3.6 Өгөгдлийн холболт

Фреймворкгүй өгөгдлийн утгыг *HTML* -н удирдлага руу түлхэх болон хэрэглэгчийн хариу үйлдлийг үйл ажиллагаа болон утгын шинэчлэл рүү шилжүүлэхийг ухаалгаар хийх хэрэгтэй. Иймэрхүү түлхэх/татах логикийк гараар бичих нь төвөгтэй, алдаа гардаг.

Angular нь өгөгдлийн холболтыг дэмждэг бөгөөд энэ нь компонентын хэсгүүдтэй загварын хэсгүүдийг тохируулахад зориулсан механизм юм. Холболтын тэмдэглэгээг *HTML* загварт нэмж өгсөнөөр хоёр талыг хэрхэн холбохыг *Angular* -т хэлж өгдөг.

Эдгээр нь өгөгдлийн холболтын синтаксын 4 хэлбэр юм.



Зураг 97. Өгөгдлийн холболтын синтаксын 4 хэлбэр

HeroListComponent жишээ загварт гурван хэлбэрийг

харуулсан байна.

```
<li>{{hero.name}}</li>
<hero-detail [hero]="selectedHero"></hero-detail>
<li (click)="selectHero(hero)"></li>
```

*Зураг 98. HeroListComponent удирдамжийн жишээ (**src/app/hero-list.component.html (binding)**)*

{{hero.name}} оруулга нь **<li>** элемент дотор компонентын **hero.name** шинж чанарын утгыг харуулна.

[hero] шинж чанарын холболт нь сонгогдсон баатрын *(selectedHero)* утгыг эцэг *HeroListComponent* -с хүү *HeroDetailComponent* -н баатрын шинж чанарт дамжуулдаг.

(click) үйл ажиллагааны холболт нь хэрэглэгч **hero.name** дээр дарах үед компонентын **selectedHero** аргыг дуудна.

Хоёр талын өгөгдлийн холболтын хамгийн чухал нь дөрөв дэх хэлбэр юм. *NgModel* удирдамжийг ашиглан нэг тэмдэглэлд шинж чанар болон үйл ажиллагааны холболтыг хослуулдаг хэлбэр юм.

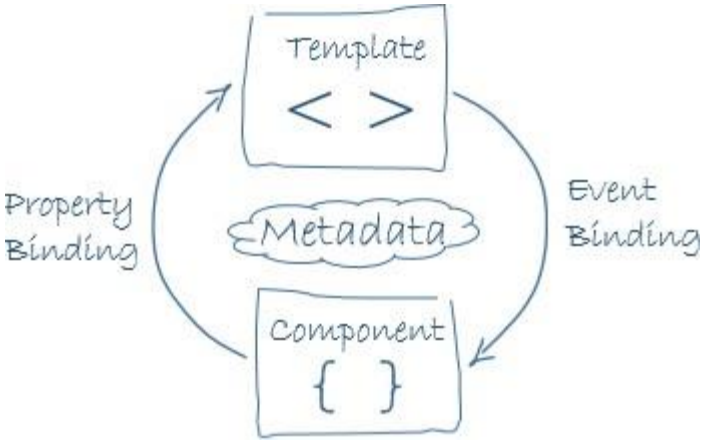
HeroDetailComponent загвараас жишээ харья.

```
<input [(ngModel)]="hero.name">
```

Зураг 99. HeroListComponent ngModel-ын жишээ (src/app/hero-detail.component.html (ngModel))

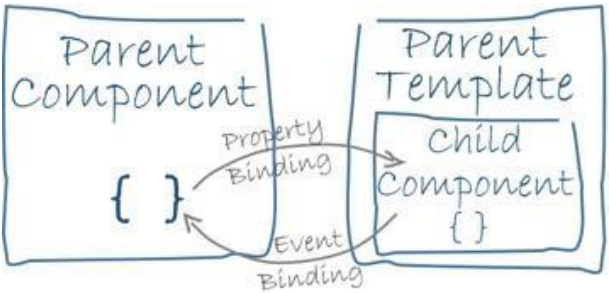
Хоёр талын холболтод өгөгдлийн шинж чанарын утгууд нь шинж чанарын холболттой адил компонентоос оролтын хайрцаг руу урсдаг. Хэрэглэгчийн өөрчлөлтүүд мөн үйл ажиллагааны холболттой адил компонент руу буцаж урсаж, шинж чанарыг хамгийн сүүлчийн утгаар дахин тохируулдаг.

Angular нь аппликэйшны компонентын модны дагуу бүх хүү компонентын эхээс бүх өгөгдлийн холболтуудыг *JavaScript* үйл ажиллагааны циклээр нэг удаа боловсруулдаг.



Зураг 100. Компонент ба загварын харилцаа

Өгөгдлийн холболт нь компонент ба загварын хоорондын харилцаанд чухал үүрэг гүйцэтгэдэг.



Зураг 101. Эцэг болон хүү компонентын харилцаа

Өгөгдлийн холболт нь эцэг болон хүү компонентуудын хоорондын харилцаанд мөн чухал үүрэг гүйцэтгэдэг.

6.3.7 Удирдамж

Angular загварууд динамик байдаг. *Angular* нь тэдгээрийг илэрхийлэхдээ удирдамжаар өгсөн зааврын дагуу *DOM* -г хувиргадаг.

Удирдамж бол *@Directive decorator* -тай класс юм. Компонент

бол удирдамж болон загвартай; *@Component decorator* бол үнэн хэрэгтээ загварт чиглэсэн онцлогуудтай өргөтгөсөн *@Directive*.

Бүтэцлэгдсэн болон атрибутын гэсэн хоёр төрлийн удирдамж байдаг. Тэдгээр нь элементийн тагд атрибутыг олгож харуулах, зарим тохиолдолд нэрээр холбох болон даалгаврыг биелүүлэх зорилготой.

Бүтэцлэгдсэн удирдамжууд нь *DOM* -н элементүүдийг нэмэх, устгах, орлуулах замаар *layout* -д өөрчлөлт оруулдаг. Жишээ загварт хоёр бүтэцлэгдсэн удирдамжийг ашигласан байна.

```
<li *ngFor="let hero of heroes"></li>
<hero-detail *ngIf="selectedHero != null"></hero-
detail>
```

Зураг 102. Бүтэцлэгдсэн удирдамж

***ngFor** нь баатруудын жагсаалтын баатар бүрд нэг **<li>** элемент гаргахыг *Angular* -т мэдэгдэж байна. ***ngIf** нь хэрвээ баатар сонгогдсон бол *HeroDetail* компонентыг агуулна.

Атрибутын удирдамж нь байгаа элементийн харагдах байдал болон зан байдалд өөрчлөлт оруулдаг. Загваруудад эдгээр нь энгийн *HTML* атрибутууд шиг харагдана.

ngModel удирдамж хоёр талын өгөгдлийн холболтыг хэрэгжүүлдэг атрибутын удирдамжийн жишээ юм. **ngModel** нь өөрчлөлтөд хариулах, дэлгэцийн утгын шинж чанарын тохиргоо байгаа элементийн зан байдлыг өөрчилдөг.

```
<input [(ngModel)]="hero.name">
```

Зураг 103. HeroDetailComponent NgModel жишээ

Angular нь цөөн тооны *layout* -н бүтцэд (жишээ нь **ngSwitch**) өөрчлөлт оруулах эсвэл *DOM* элементүүд болон компонентуудын (жишээ нь **ngStyle** болон **ngClass**) гадаад

байдлыг өөрчлөх зэрэг цөөн тооны удирдамжтай.

Мэдээж та өөрийн удирдамжийг бичиж болно. *HeroListComponent* шиг компонентууд тусгай удирдамжийн нэг төрөл юм.

6.3.8 Сервис

Сервис бол ямар ч утга, функц эсвэл аппликэйшны шаардлагыг багтаасан өргөн хүрээтэй ангилал юм. Бараг ямар ч зүйл сервис байж чадна. Ерөнхийдөө сервис бол нарийн сайн тодорхойлсон зорилготой класс бую тодорхой нэг зүйлийг сайн хийдэг байх ёстой. Жишээлбэл:

- Logging service
- Data service
- Message bus
- Tax calculator
- Application configuration

Angular -т сервисүүдийн тухай онцлог бөгөөд тодорхойлолт байхгүй. Энд сервисийн суурь класс болон сервисийг бүртгүүлэх зүйл байхгүй.

Гэсэн хэдий ч сервисүүд нь ямар ч *Angular* аппликэйшны үндсэн хэсэгт байдаг. Компонент бүр сервисүүдийн томоохон хэрэглэгчид юм. Вэб хөтөчийн консол руу бичих сервис классын жишээ энд байна.

```
class Logger {
  void log(Object msg) => window.console.log(msg);
  void error(Object msg) =>
window.console.error(msg);
  void warn(Object msg) =>
window.console.warn(msg);
}
```

Зураг 104. Logger service жишээ

HeroService баатруудыг дуудах *Promise* -г ашиглана. *HeroService* нь өөр нэг хамтын ажиллагааг зохицуулдаг *BackendService* болон *Logger* сервисээс хамаарна.

```
class HeroService {
  final BackendService _backendService;
  final Logger _logger;
  final heroes = <Hero>[];

  HeroService(this._logger, this._backendService);

  List<Hero> getHeroes() {
    _backendService.getAll(Hero).then((heroes) {
      _logger.log('Fetched ${heroes.length}
heroes.');
      this.heroes.addAll(heroes as List<Hero>); //
fill cache
    });
    return heroes;
  }
}
```

Зураг 105. HeroService жишээ

Сервисүүд хаана ч байдаг. Компонентын классууд нь тулгуур болж байх ёстой. Тэд серверээс өгөгдөл дуудах, хэрэглэгчийн оролтыг баталгаажуулах эсвэл консол руу шууд бичдэггүй. Тэд ийм ажлуудыг сервисд шилжүүлдэг. Компонентуудын ажил бол зөвхөн хэрэглэгчтэй харилцах юм. Энэ нь харагдац (загвараар илэрхийлэгддэг) болон аппликэйшн логикийн хооронд дамждаг. Сайн компонент нь өгөгдлийн холболтод зориулсан аргууд болон шинж чанаруудыг харуулдаг. Энэ нь ердийн биш (*non-trivial*) бүх зүйлсийг сервис төлөөлдөг.

6.3.9 Хамааралтай ролт

Хамааралтай ролт нь бүрэн хэлбэржсэн хамаарлуудыг

шаарддаг классын шинэ тохиолдолыг дэмжих нэг арга юм. Ихэнх хамаарлууд нь сервисүүд юм. *Angular* нь сервисүүд нь шаарддаг шинэ компонентуудыг хангахын тулд хамааралтай оролтыг ашигладаг.

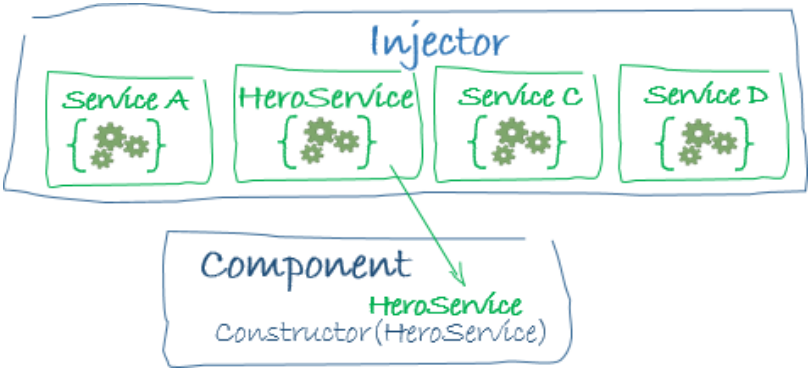
Angular нь компонентын байгуулагчийн параметрүүдийн төрлүүдийг харсанаар тухайн компонентэд аль сервис хэрэгтэйг хэлж чадна. Жишээ нь таны *HeroListComponent* -д *HeroService* хэрэгтэй.

```
final HeroService _heroService;
HeroListComponent(this._heroService);
```

Зураг 106. HeroListComponent constructor

Angular нь компонент үүсгэхдээ хамгийн түрүүнд компонентыг шаарддаг сервисүүдийн *injector* -г асуудаг. *Injector* нь өмнө үүссэн сервисийн тохиолдлуудыг агуулж байдаг. Хэрэв хүссэн сервисийн тохиолдол контайнерт байхгүй бол *injector* нь *Angular* -т сервисийг буцаахаас өмнө контайнерт нэгийг үүсгээд нэмдэг. Бүх хүсэлт илгээсэн сервисүүд шийдвэрлэгдсэн болон буцаагдсан үед *Angular* нь аргументууд шиг сервисүүдтэй компонентын байгуулагчийг дуудаж чадна. Энэ бол хамааралтай ороолт юм.

HeroService оролтын процесс нь дараах шиг харагдана.



Зураг 107. Injector майлбар

Хэрэв *injector* -т *HeroService* байхгүй бол үүнийг хэрхэн үүсгэх вэ? Өөрөөр хэлбэл, *injector* -той *HeroService* -н хангагчийг өмнө бүртгүүлсэн байх ёстой. Хангагч нь сервисийг үүсгэж, буцааж чаддаг зүйл юм. Ихэвчлэн сервис класс өөрөө байдаг. Компонентууд дахь эсвэл модулиуд дахь хангагчуудыг бүртгэж чадна.

Хаа сайгүй байдаг сервисүүдийн байршлыг олохын тулд хангагчийг **`root`** модульд нэмэх хэрэгтэй.

```
@Component(
    // ...
    providers: const [BackendService, HeroService,
Logger],
  )
  class AppComponent {}
```

Зураг 108. Хангагчдыг бүртгэх

Өөр нэг арга нь *@Component* -н мета өгөгдлийн хангагчийн шинж чанарт компонентын түвшинг бүртгэнэ.

```
@Component({
    selector:       'app-hero-list',
    templateUrl:    './hero-list.component.html',
    providers:      [HeroService]
})
```

Зураг 109. @Component decorator жишээ

Компонентын түвшинг бүртгэх гэдэг нь тухайн компонентын шинэ тохиолдол тус бүрт сервисийн шинэ тохиолдлыг авна гэсэн үг.

Хамааралтай оролтын талаар санамж: Хамааралтай оролт нь *Angular* фреймворкын хүрээнд болон хаа сайгүй хэрэглэгддэг.

- *Injector* бол гол механизм нь юм.
- *Injector* нь үүссэн сервисийн тохиолдлын контайнерыг

агуулж байдаг.

- *Injector* нь хангагчаас шинэ сервисийн тохиолдлыг үүсгэж чаддаг.
- Хангагч бол сервис үүсгэх заавар юм.
- *Injector* -ууд нь хангагчийг бүртгэдэг.

Дээрх агуулга нь *Angular* аппликэйшны суурь ойлголт юм. Энэ нь цааш хөгжүүлэлт хийхэд хангалттай биш боловч бүх зүйлийг мэддэг байх шаардлагагүй.

Дараах зарим чухал *Angular* -н онцлогийг товч дурдая.

- *Animations*: *Animate* компонент нь *Angular* -н дүрсийг хөдөлгөөнд оруулах сан.
- *Change detection*: Өөрчлөлтийг илрүүлэгч нь *Angular* тухайн компонентын шинж чанарын утга өөрчлөгдөх үед дэлгэцийг шинэчлэх болон өөрчлөлтийг илрүүлэгчийн стратекийн асинхрон үйл ажиллагааг тасалдуулах, ажиллуулах бүсчлэлийг хэрхэн ашиглах зэргийн шийдлийг баримтжуулж өгдөг.
- *Event*: Үйл ажиллагааг нийтлэх, нэгтгэхэд зориулагдсан механизм үйл ажиллагаагаар хангагдсан компонент болон сервисүүдийг хэрхэн өргөтгөхийг баримтжуулж өгдөг.
- *Forms*: *HTML* -д суурилсан буруу өгөгдлийг шалгах болон баталгаажуулалтай цогц өгөгдөл оруулах хувирбаруудыг дэмждэг.
- *HTTP*: Өгөгдөл авах, өгөгдөл хадгалах болон *HTTP* клиентээс сервер талын үйл ажиллагааг дуудах сервертэй харьцдаг.
- *Lifecycle hooks*: Интерфэйсийг хэрэгжүүлсэнээр компонентыг үүсгэхээс устгах хүртэл компонентын амьдралын мөчлөг дэх агшин бүрийг дарж байдаг.
- *Pipe*: Дэлгэцэнд зориулсан утгуудыг шилжүүлэгчээр *UX* -г сайжруулахын тулд таны загварт пайпуудыг хэрэглэдэг.

- *Router*: Клиент аппликэйшн дотор хуудаснаас хуудас руу чиглэдэг. Хэзээ ч вэб хөтөчөөс явдаггүй.
- *Testing*: *Testing Platform* ашиглан *Angular* фреймворктой харьцахдаа таны аппликэйшны хэсгүүдэд нэгжийн тест хийдэг.

6.4. Технологийн хэрэглээ

Angular фреймворкын гол зорилго нь вэб аппликэйшны *front-end* буюу хэрэглэгчийн талын интерфейсыг хөгжүүлэхэд оршино.

Энэ технологийг ашиглах гол системүүд бол нэг хуудаст аппликэйшнууд, вэб сайтууд, томоохон системүүдийн модулиудыг зохион байгуулах болон бодит хугацаанд өгөгдлийг дамжуулах системүүд юм.

Angular нь *front-end* фреймворк бөгөөд *RESTful API* ашиглан бүх төрлийн *back-end* технологиудтай хамтарч ажиллах боломжтой ба тэдгээрээр дамжуулан өөр бусад системүүдтэй ч холбогдож ажиллаж чадна.

Бүх төрлийн төхөөрөмжинд зориулсан нэг хуудаст аппликэйшн, вэб сайт болон бодит хугацаанд ажиллах ёстой системүүдийг хөгжүүлэх боломжтой.

BACKBONE JAVASCRIPT

7. BackboneJS

Jeremy Ashkenas анх энэхүү фреймворкыг хөгжүүлж, 2010 оны 10 сарын 13 -нд танилцуулжээ [4]. *BackboneJS* нь *Lightweight JavaScript library* -г дэмжин клиент талын аппликэйшнийг вэб хөтөч дээр ажиллуулах хөгжүүлэлтийг дэмждэг. *BackboneJS* нь вэб аппликэйшнийг илүү хурдан ажиллах боломжийг бүрдүүлж, тодорхойлогдсон *models, views, collections* -с сонгон ашиглах, сервер дээр заавал бүх үйлдлийг хийх шаардлагагүй болсон. *JavaScript* нь клиент тал дээр ажилладаг. *RESTful JSON interface* дээрх *API* -тай холбогдон ажилладаг. Кодыг *JavaScript* хэл дээр бичиж *HTML* болон тухайн серверийн дэмждэг файлтай холбож ажиллуулдаг. Энэ фрэймворкын давуу талыг одоо дурдая.

- *BackboneJS* нь аппликэйшний урд хэсэг буюу харагдах хэсгийг *JavaScript* -н функцийг ашигласнаар маш хялбархан аргаар хөгжүүлэх боломжийг олгодог.
- Хэрэглэгчийн талын вэб аппликэйшнд **`models, views, events, routers болон collections`** -р хангаж өгдөг.
- *Model* буюу өгөгдөл солигдох үед автоматаар аппликэйшний *HTML* завсарлагдана.
- Энэ нь нээлтэй эх бөгөөд ба 100 гаруй идэвхитэй нэмэлт үйлчилгээг агуулдаг.
- Энгийн кодыг системтэйгээр маш эмх цэгцтэй болгодог.
- *BackboneJS* нь *Underscore JS* болон *Jquery* -тэй хамтарч ажилладаг.
- *BackboneJS* нь хөгжүүлэгчид клиент талын вэб програм буюу гар утасны програмуудыг тодорхой форматаар үүсгэх боломжийг олгодог.

Харин дараах сул талуудыг ажиглаж хөгжүүлэлтэндээ анхаарах хэрэгтэй.

- *BackboneJS* сан нь ганцхан хуудсыг хөгжүүлэх боломжтой.
- *Backbone* нь *light-weight* санг ашигладаг ба *front-end* талыг хөгжүүлэхэд сайн ч энэ *JavaScript* -н сан бөгөөд back-end талдаа хэрэглэх зориулалттай юм.

7.1. Үйлдлийн системийн шаардлага

BackboneJS нь вэб аппликэйшнд зориулсагдсан учир *Windows, Unix, Linux, MacOS* үйлдлийн системүүд дээр хөгжүүлэх боломжтой. *RESTful Json interface* дээрх *API* -тай холбогдон ажилладаг. Сан нь *model view presenter (MVP)* аппликэйшн дээр суурилсан.

7.2. Програмчлалын хэлний шаардлага

Вэб хөгжүүлэх гэж байгаа учир *HTML, CSS* болон *JavaScript* хэлний мэдэгдэхүүнтэй байх шаардлагатай. Хэрвээ анхан шатны програмист бол тухайн сангийн талаар мэдээлэл авах хангалттай олон эх үүсвэр байдаг. Вэб аппликэйшнд олон *JavaScript* -г оруулан ажиллуулах үед *HTML* DOM руу өгөгдлийн оролтыг сурах тухай бодох хэрэггүй. Учир нь таны үүсгэсэн *JavaScript* аппликэйшний ард *Jquery* -н дахин дуудах, сонгох болон *HTML UI* -н мэдээллийг сервер лүү синк хийх болон *UI* -г дахин засварлах боломжуудтай. Үүнээс гадна клиент талд илүү их тусламжуудаар хангаж өгсөн.

7.3. Суулгах програмчлалын хэрэгслүүд

Мэдээж програм хангамж шинээр үүсгэх эсвэл хөгжүүлэх гэж байгаа учир тухайн хөгжүүлэх зүйлдээ тохирох *tool* буюу хэрэгслүүдийг суулгах хэрэгтэй. Энэ нь та өөрийн програм хангамжийн орчноо бүрдүүлж байгаа хэрэг юм. Бидний авч үзэж байгаа сан нь вэб аппликэйшнд зориулагдсан болохоор вэб аппликэйшн хөгжүүлж болох ямар ч хэрэгслийг сонгон авж ашиглах боломжтой. Жишээлбэл: *sublime tool* дээр маш олон төрлийн хэл дээр код бичих боломжтой. Нэг үгээр хэлбэл

бидний байнга хэрэглэдэг *notepad* -тай адилхан. *Notepad* -аас давуу тал гэвэл зарим түлхүүр үгнүүд нь өнгөөр харагдах, алдаа гарсан тохиолдолд тухайн алдаатай хэсгийн өнгө өөр харагддаггэх мэт. Мөн тухайн програмчлалын хэлний түлхүүр үгүүдийн гүйцээлтийг гаргаж ирдэг. Энэ нь кодыг хурдан бичих мартсан бол зөв буруу зэргийг харах боломжтой байдаг. Үүнээс гадна маш олон хэрэгсэл байдаг. *Atom, Notepad++* гэх мэт. Та өөрийн хэрэгслээ сонгосон бол одоо *BackboneJS, Underscore JS, Json*, болон *JQuery* татах хэрэгтэй. Энэ нь бидний хэрэглэх гэж байгаа *JavaScript* -н сан юм. Өөрийн компьютерийн аль нэг хэсэгт өөрийн вэбийн хавтсыг үүсгэж түүн дотор татсан файлуудыг хуулна. Мөн *HTML* файл үүсгэж тухайн файлуудыг хуудастай холбож өгнө. Ингэснээр та энэхүү сангийн онцлог давуу талуудыг хэрэглэх боломжтой болно.

7.4. Өгөгдлийн сангийн шаардлага

Өгөгдлийн сангийн хувьд дурийн өгөгдлийн сантай холбогдох боломжтой ба тухайн холбогдож байгаа өгөгдлийн сантай холбогдох серверийг ашиглах хэрэгтэй. Жишээлбэл: *Mysql* дээр өгөгдлийн санг байгуулаад холбохыг хүсвэл *Apache* серверээр дамжуулан холбогдож дунд нь *PHP* файлыг тодорхойлж өгөх хэрэгтэй. Харин *BackboneJS* нь түүнийг *JSON* руу хөрвүүлж уншаад *HTML DOM* руу өгөгдлийг харуулах байдлаар ажилладаг. *BackboneJS* -г *NodeJS JavaScript* -н фрэймворктой ажиллахад илүү хялбар бөгөөд *MongoDB* ашиглан өгөгдлийг маш хялбархан байдлаар хүлээн авч ажиллах болно. Харин бусад серверүүд тэдэнд тохирох өгөгдлийн сантай холбогдож ажиллах боломжтой ч тэр нь тийм ч тохиромжтой бус. Харин өөрийн фрэймворк өгөгдлийн санг ашигласнаар илүү хялбар програмчлах боломжтой юм.

7.5. Суурилуулах заавар

Сервер тал: Дурын вэб аппликэйшнд зориулсан сервер дээр өөрийн вэб аппликэйшнийг үүсгэнэ. Тухайн хэрэглэгчийн цонхонд *BackboneJS, Underscore JS, Json2, JQuery* болон өөрийн

custom JS файлыг үүсгэж холбож өгөх хэрэгтэй. Ингэснээр тухайн сан сервер талд ажиллуулах боломжтой болно. Энэхүү сан нь сервертэй дараах `гурван` функцийн тусламжтайгаар ажилладаг.

- fetch() - GET()
- save() - POST/PUT
- delete() - remove

```
<!DOCTYPE html>
<html>
<head>
<title>Backbone JS</title>
<link rel = "stylesheet" type = "text/css" href =
"css/bootstrap-min.css">
</head>
<body>
<div class = "container">
<h1>Backbone JS tutorial - Demo</h1>
<table class = "table">
<thead>
    <tr>
        <th>Student code</th>
        <th>Student name</th>
        <th>Major</th>
        <th>GPA</th>
    </tr>
    <tr>
        <td><input class = "form-control code-in-
put"></td>
        <td><input class = "form-control name-in-
put"></td>
        <td><input class = "form-control major-in-
put"></td>
        <td><input class = "form-control gpa-in-
put"></td>
```

```
        <td><button class = "btn btn-primary add-
student">Add</button></td></tr>
</thead>
<tbody class = "student-list"></tbody>
</table>
</div>
<script type = "text/template" class = "student-
list-template">
<td><span class = "code"><%= code %></span></td>
<td><span class = "name"><%= name %></span></td>
<td><span class = "major"><%= major %></span></td>
<td><span class = "gpa"><%= gpa %></span></td>
<td>
<button class = "btn btn-warning edit-
blog">Edit</button>
<button class = "btn btn-danger delete-
blog">Delete</button>
<button class = "btn btn-success update-blog"
style = "display:none">Update</button>
<button class = "btn btn-danger cancel-blog" style
= "display:none">Cancel</button>
</td>
</script>
<div id = "unique"></div>
<script type = "text/javascript" src = "js/jquery-
min.js"></script>
<script type = "text/javascript" src =
"js/undercore-min.js"></script>
<script type = "text/javascript" src =
"js/backbone-min.js"></script>
<script type = "text/javascript" src =
"js/myJavascript.js"></script>
</body>
</html>
```

Зураг 110. Энгийн html хуудас

Энэ функцүүдийг ашиглан серверээс мэдээлэл унших, засвар хийх, устгах, синк хийх үйлдлүүдийг хийх боломжтой болно.

```
Backbone.Model.prototype.idAttribute = '_id';

var Student = Backbone.Model.extend({
    defaults:{
        code: '',
        name: '',
        major: '',
        gpa: ''
    }
});

var Students = Backbone.Collection.extend({
    url: 'http://localhost:3000/api/blogs'
});

var class_1 = new Students();

var StudentView = Backbone.View.extend({
    student: new Student(),
    tagName: 'tr',
    initialize: function(){
        this.template = _.template($('.student-
list-template').html());
    },

    events: {
        'click .edit-blog': 'edit',
        'click .update-blog': 'update',
        'click .cancel-blog': 'cancel',
        'click .delete-blog': 'delete',
    },
```

```
edit: function(){
    $('.edit-blog').hide();
    $('.delete-blog').hide();
    this.$('.update-blog').show();
    this.$('.cancel-blog').show();

    var code = this.$('.code').html();
    var name = this.$('.name').html();
    var major = this.$('.major').html();
    var gpa = this.$('.gpa').html();

this.$('.code').html('input type="text" class =
"form-control code-update" value= ' + code + '>');
this.$('.name').html('input type="text" class =
"form-control name-update" value= ' + name + '>');
this.$('.major').html('input type="text" class=
"form-control major-update" value=' +major + '>');
this.$('.gpa').html('input type = "text" class =
"form-control gpa-update" value='+gpa + '>');
},

update: function(){
this.model.set('code', $('.code-update').val());
this.model.set('name', $('.name-update').val());
this.model.set('major', $('.major-update').val());
this.model.set('gpa', $('.gpa-update').val());
    }
})
```

Зураг 111. Энгийн custom js буюу myJavascript.js файл

Клиент дээр: Тухайн сангийн *models, events, collection* -ийг ашигласнаар өөрийн *models, events, collections* -ийг тодорхойлж өгдөг. Энэхүү сан нь энгийн вэб аппликэйшнийг бодвол хурдан ажиллагаатайгаас гадна сервер дээрх үйлдлүүдийг хийх биш клиент дээр үйлдлүүдийг хийж боловсруулаад боловсруулсан

өгөгдлөө сервер лүү синк хийж өгдөг. Ингэснээр сервер дээр хэт их ачаалал ирэхгүйгээс гадна маш хурдан ажиллах боломжтой. Дараах жишээг харна уу.

```
var express = require('express');
var bodyParser = require('body-parser');
var mongoose = require('mongoose');
var router = express.Router();

mongoose.connect('mongodb://127.0.0.1:27017/blogro
ll', function(err, db){
    if(err){
        throw err;}
    else {
        console.log("Connected.");
    }
});

var Schema = mongoose.Schema;

var BlogSchema = new Schema({
    code: String;
    name: String;
    major: String;
    gpa: String
});

mongoose.model('Blog', BlogSchema);

var Blog = mongoose.model('Blog');
var app = express();

app.use(express.static(_dirname + "/public"));
app.use(bodyParser.json());
```

```
//Замчлал
app.get('/api/blogs', function(req, res){
    Blog.find(function(err, docs){
        docs.forEach(function(item){
            console.log('Recieved a GET request
for_id: '+ item._id);
        });
        res.send(docs);
    });
});

app.post('/api/blogs', function(req, res){
    console.log('Received a POST request');
    for(var key in req.body){
        console.log(key + ': ' + req.body[key]);
    }
    var student = new Blog(req.body);
    student.save(function(err, doc){
        res.send(doc);
    });
});
```

Зураг 112. Жишээнд ашиглагдаж буй server.js файл

7.6. Програмчлалын хэрэгсэлтэй холбох

Өөрийн үүсгэсэн вэб аппликэйшний хавтсыг өөрийн хэрэгсэл дээрээ оруулснаар таньд илүү хялбар байдлыг бүрдүүлж өгдөг. *Sublime* болон *Atom* дээр өөрийн хавтсаа зөөж оруулах хэрэгтэй. Ингэснээр та вэб аппликэйшний хавтсаа бүрэн ашиглаж болно. Татаж авсан файлыг өөрийн *HTML* хуудастайгаа холбож өгөхийн тулд дараах мөр кодыг ашиглана.

```
hline <script type="text/JavaScript"
src="PATH> </script>
hline
```

Энэ мөр кодыг ашиглан *HTML* хуудсыг *JS* файлтай холбож өгдөг.

PATH нь таны *JS* файлыг заах зам юм. Ингэснээр таны *HTML* хуудас *JavaScript* файлуудтай холбогдож өгнө. Та одоо өөрийн *JavaScript* файлаа үүсгээд *HTML* -н *DOM* -г ашиглан хүссэн байдлаар засварлах өөрчлөх, хэлбэржүүлэх гэх мэт зүйлүүдийг хийх боломжтой.

7.7. Функц үйлдлүүдийн хэрэглээ

7.7.1 Model - Загвар

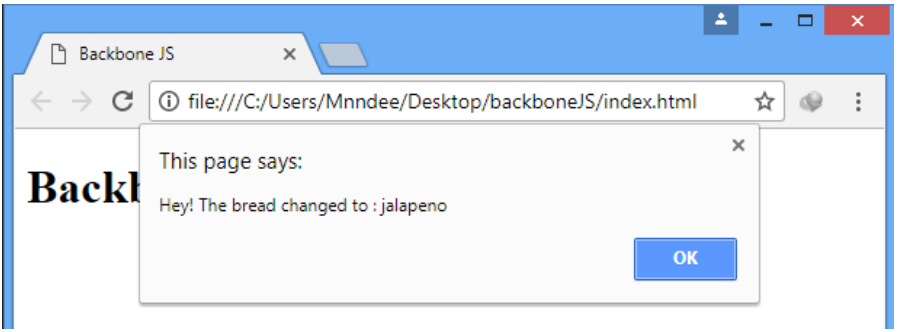
Модел солигдох үзэгдлийг энгийн жишээгээр харуулья.

```
<!DOCTYPE html>
<html>
<head>
    <title>Backbone JS</title>
</head>
<body>
    <h1>Backbone JS tutorial 1</h1>

<script type = "text/javascript" src = "js/jquery-
min.js"></script>
<script type = "text/javascript" src =
"js/undercore-min.js"></script>
<script type = "text/javascript" src =
"js/backbone-min.js"></script>
<script type = "text/javascript" src =
"js/myJavascript.js"></script>
</body>
</html>
```

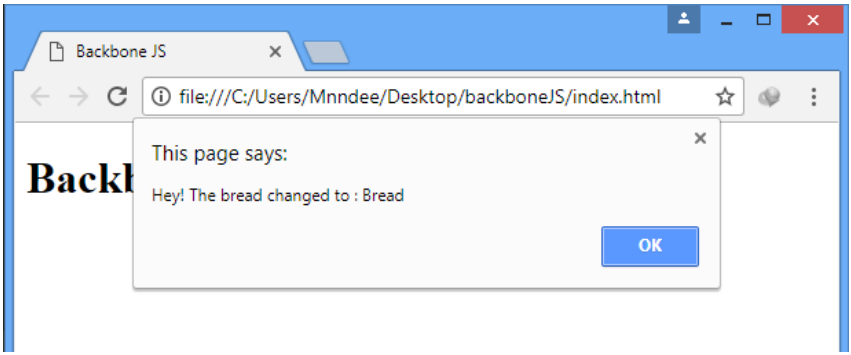
Зураг 113. Энгийн html хуудас

```
SandwichModel = Backbone.Model.extend({
    defaults: {
        bread: "улаан буудайн гурил"
    },
    initialize: function(){
        this.bindEvents();
    },
    bindEvents: function(){
        this.on("Өөрчлөлт: талх", function(model){
            var my_new_bread = model.get("талх");
            alert("Талхыг " +my_new_bread + "
талхаар өөрчиллөө.");
        });
    }
});
$(document).ready(function(){
    var my_sandwich = new SandwichModel();
    my_sandwich.set({bread: "Чанамал"});
    my_sandwich.set({bread: "Талх"});
});
```

Зураг 114. Энгийн хэрэглэгчийн тодорхойлсон JavaScript файл буюу myJavascript.js



Зураг 115. Үр дүн



Зураг 116. Үр дүн

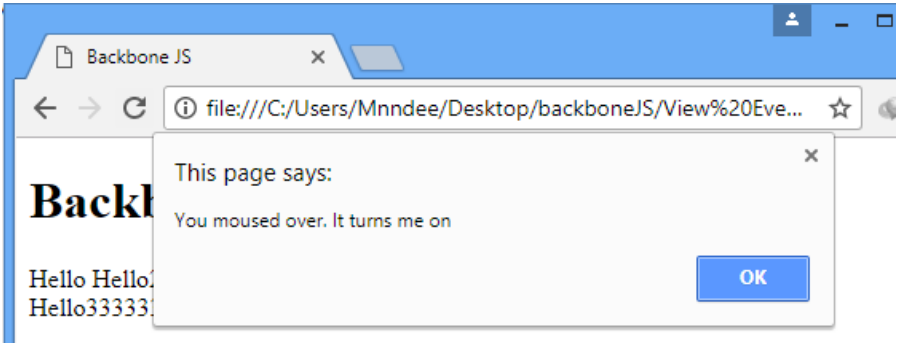
7.7.2 View - Харагдах хэлбэр

```
<!DOCTYPE html>
<html>
<head>
    <title>Backbone JS</title>
</head>
<body>
    <h1>Backbone JS tutorial View</h1>
    <div id = "unique"></div>
<script type = "text/javascript" src = "js/jquery-
min.js"></script>
<script type = "text/javascript" src =
"js/undercore-min.js"></script>
<script type = "text/javascript" src =
"js/backbone-min.js"></script>
<script type = "text/javascript" src =
"js/myJavascript.js"></script>
<script type = "text/template" id = "myTemplate">
    <span class = "Hello">Сайн байна уу</span>
</script>
</body>
</html>
```

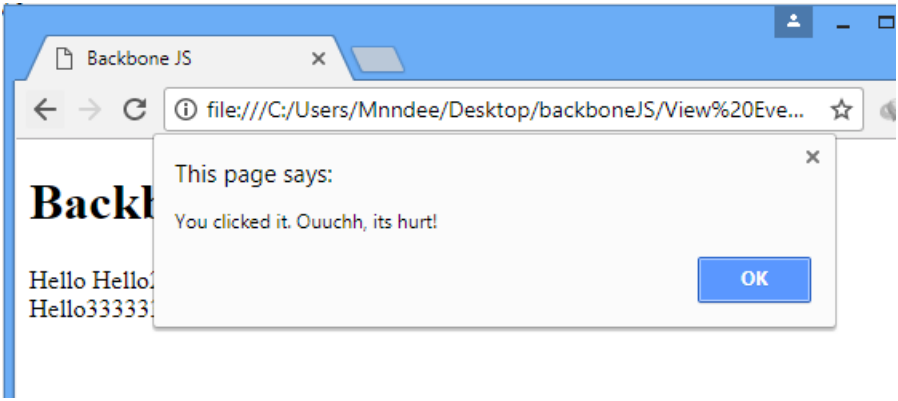
Зураг 117. html хуудсанд тэмплэйт ашиглах

```
TheView = Backbone.View.extend({
    initialize: function(){
        this.render();
    },
    render: function(){
        var template =
_.template($("#myTemplate").html(),{});
        this.$el.html(template);
    },
    events: {
        "click": "clicked",
        "mouseover span.hello": "mouseover"
    },
    clicked: function(){
        alert("You clicked it!");
    },
    mouseover: function(){
        alert("You moused over.");
    }
});
$(document).ready(function(){
    var aView = new TheView({el: $("#unique")});
});
```

Зураг 118. Энгийн JavaScript хуудас



Зураг 119. Хулгана дээгүүр өнгөрөх үзэгдэл



Зураг 120. Товших үзэгдэл

7.8. Технологийн хэрэглээ

Вэб хөгжүүлэх боломжтой бүх програмчлалын хэлнүүдтэй хамтарч ажиллах боломжтой. *JavaScript, PHP* -ийн фрэймворктой хамтран ажиллах боломжтой. *BackboneJS* нь *Underscore* дээр суурилсан *JQuery* -г ашигладаг сан болно. Өөрийн хүссэн вэб хөгжүүлэх хэл дээр вэб аппликэйшнээ бичих үедээ *BackboneJS, Underscore, JQuery* зэргийг татаж өөрийн вэб аппликэйшнийг холбож өгснөөр та ашиглах боломжтой болно. Бүх вэб хөгжүүлэх боломжтой хэлүүдтэй зохицож ажиллах боломжтой боловч *JavaScript* -н фрэймворкуудийг сонговол илүү хөгжүүлэхэд хялбар.

Хангалттай загвар, загварын цуглуулга, харагдах цонх, замчлал болон үзэгдлүүдээр хангагдсан байдаг учир цөөхөн цонхтой, олон үзэгдэлтэй, нэг цонхонд хэд хэдэн загварыг ашигладаг аппликэйшнийг хөгжүүлэхэд ашигласнаар давуу талуудыг нь харах боломжтой. Бас энгийн аппликэйшнээс хэд дахин хурдан ажиллахыг мэдэрч болно.

KNOCKOUT JAVASCRIPT

8. KnockoutJS

Орчин үеийн вэб хуудасны хөгжүүлэлтийн чиг хандлага бол бага нөөц зарцуулах, тооцооллыг хэрэглэгч талд шийдвэрлэх, хэрэглэгчдэд ашиглахад хурдан шуурхай, тааламжтай интерфэйстэй байхыг зорих болсон. Иймээс бидэнд илүү хялбар, хурдан шуурхай, хэрэглэгчийн интерфейсийг зохиомжлох боломжтой хэлний шаардлага гарч ирсэн.

KnockoutJS нь үндсэндээ *MVVM* загвар дээр үндэслэсэн *JavaScript* дээр бичигдсэн фреймворк юм. *KnockoutJS* фреймворк нь өгөгдөлд суурилсан интерфэйсүүдийг хялбар аргаар зохицуулдаг. Энэ нь бусдаас хараат бус байдлыг бий болгодог. *MVVM* загвар нь *Model, View* болон *View, Model* -ийг тусад нь салгадаг.

KnockoutJS -г Steve Sanderson зохиосон бөгөөд 2010 оны 7-р сарын 5 -нд анх удаа олон нийтэд танилцуулжээ. Одоогоор 3.5 хувилбар гараад байна. Хөгжүүлэлтийн төлөв идэвхтэй бөгөөд үндсэн хуудас knockoutjs.com, github дээр github.com/knockout/knockout.

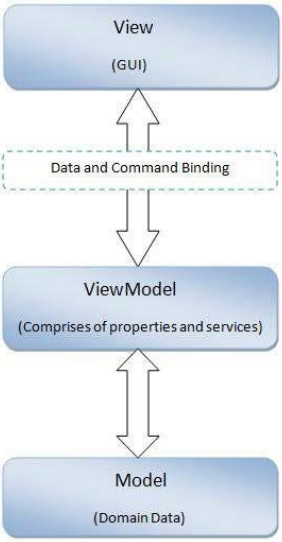
8.1. Үйлдлийн системийн шаардлага

Уг технологи *JavaScript* -н адил үйлдлийн систем хамаарахгүй ажиллана. Ямар ч төрлийн үйлдлийн систем дээр ажиллана. Энэ нь хөгжүүлэгчдэд илүү амар болох бөгөөд түгээмэл ашиглагддаг. Энэ тухайн хэл удаан оршин тогтнох үндэс болдог.

8.2. Програмчлалын хэлний шаардлага

Knockout -г ашиглахын тулд үндсэн *HTML* болон *JavaScript* -ийн мэдлэг хэрэгтэй. Үүнээс гадна, уг хэлний архитектур болох MVVM -г мэддэг байх шаардлагатай.

8.2.1 MVVM архитектур



Зураг 121. MVVM Архитектур

Model-View View- Model (MVVM) нь програм хангамжийн хэрэглээг хөгжүүлэх архитектурын загвар юм. *John Gossman* 2005 онд *Microsoft MVVM* архитектурыг боловсруулсан. Энэ загвар нь *Model-View-Controller (MVC)* загвараас гаралтай. *MVVM* -ийн давуу тал нь хэрэглэгчийн програмыг график интерфэйсээс салгаж гаргадаг.

8.3. Суурилуулах заавар

KnockoutJS -ийг ашиглахад маш хялбар байдаг. Дараах байдлаар холбогдож болно.

Зураг 122. Суулгах файл

KnockoutJS -г татаж авахын тулд албан ёсны өөрийн сайт руу

орж татна.

/http://knockoutjs.com/downloads/index.html/

Дараах кодоор дамжуулан хадгалж авсан файлаа дуудаж өгнө.

```
<script type = "text/JavaScript" src = "knockout-
3.4.2.js"></script>
```

Content Delivery Networks (CDN) -ээс *KnockoutJS* санг авч болно. *Microsoft Ajax CDN* -ээс *KnockoutJS* -ийн санг дараах байдлаар уншина.

```
<script type = "text/JavaScript" src =
"https://ajax.aspnetcdn.com/ajax/knockout/knockout
-3.4.2.js"></script>
```

Knockout-server-side-validation нь *JavaScript* дахь серверийн алдааны мэдээллийг үзүүлэх энгийн *JavaScript* сан юм. Анхдагчаар, серверийн зурвас бүр элементийн алдааны хажууд харуулагддаг. Энэ үйлдлийг өөрчилж болно. Энэ мөчид сан *ASP.NET MVC* сервер талын баталгаажуулалтыг хялбархан хийх боломжийг олгодог бөгөөд хэрэв шаардлагатай бол зарим нэг өөр орчинд ашиглах боломжтой болно.

Хэрэв *AJAX* ашиглаж байхад загвар нь бүтэлгүйтсэн тохиолдолд клиент дээр сервер талын баталгаажуулалтын алдааг харуулахдаа доорх зүйлсийг хийх хэрэгтэй. Үүнд:

- *Knockout* холболтын замыг өөрчлөх
- Сервер дээр баталгаажуулах мессежийг удирдах замыг өөрчлөх
- Клиент дээрхи баталгаажуулах мессежийг хянах замыг өөрчлөх

Клиент тал:

```
ko.applyBindings(viewModel,    rootNode);
ko.applyBindingsWithValidation(viewModel,
rootNode);
```

Сервер талд *ASP.NET MVC* хэсэгт энгийн өргөтгөл нэмэх хэрэгтэй.

```
public static class ModelStateExtentions{

public static JsonResult ToKoJsonResult(this
ModelStateDictionary modelState)
{
return new JsonResult(){
      Data = new {KoValid = modelState.IsValid,
ModelState}
}
}
public static object ToKoJson(this
ModelStateDictionary modelState)
{
return modelState.where(x=>x.value.errors.count >
0).select(ms=>new
{
    key = ms.key,
    value = ms.value.errors.first().errorMessage
      }).toArray();
   }
}
```

Серверийн баталгаажуулалтын алдааг зохицуулж өөрчлөх хэрэгтэй. Эхлээд ASP.NET MVC дээр шинж чанарыг тодорхойлох хэрэгтэй.

```
public class KoJsonValidateAttribute:
ActionFilterAttribute
{
protected bool IgnoreValidation;
public KoJsonValidateAttribute(bool
IgnoreValidation = false)
```

```
{this.IgnoreValidation = IgnoreValidation;}
public override void
OnActionExecuting(OnActionExecutingContext
filterContext)
{if(IgnoreValidation ||
!filterContext.httpContext.Request.isAjaxRequest()
)
{return;}
var modelState =
filterContext.Controller.ViewData.modelState;
if(!modelState.isValid)
{ filterContext.httpContext.Response.clear();

filterContext.httpContext.Response.StatusDescripti
on = "Validation error";
filterContext.Result =modelState.ToKoJsonResult();

filterContext.Result.ExecuteResult(filterContext.C
ontroller.ControllerContex);
    filterContext.httpContext.clearError();
    filterContext.httpContext.Response.end();
      }
    }
}
```

Өгөгдлийг баталгаажуулах боломжтой шинж чанарыг нэмэх нь:

```
[HttpPost, KoJsonValidate]
public ActionResult Save(MyViewModel model)
{
    return Json(new {redirect =
url.action("RedirectAction")});
}
```

Клиент талд өгөгдлийг серверээс буцаах үед кодын мөр нэмэх хэрэгтэй.

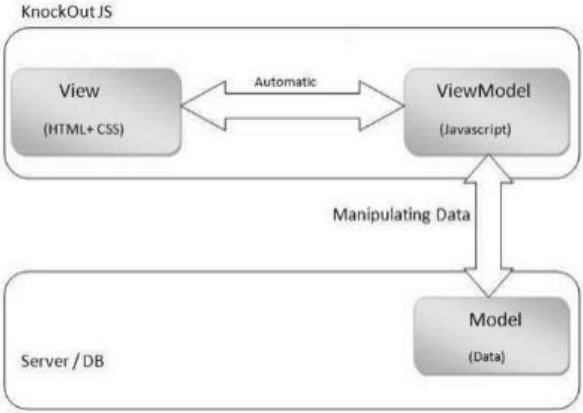
```
ko.serverSideValidator.validateModel(viewModel,
data);
```

`viewModel` загвар бөгөөд өгөгдөл нь *AJAX* дуудлагаас серверээс ирсэн өгөгдөл юм.

KnockoutJS -г ашигласны давуу талуудыг одоо дурдая.

- *KnockoutJS* сангууд нь өгөгдөлд суурилсан интерфэйсийг хялбар аргаар зохицуулдаг. Өөрөөр хэлбэл *JavaScript* объектод шинэчлэх *UI* -ийг үүсгэж болно.
- Энэ бол *JavaScript* сан бөгөөд ямар ч вэбийн орчинд ажилладаг. Энэ нь *JQuery* -ийн орлуулалт биш харин ухаалаг функцийг бий болгох нэмэлт хэрэгсэл болж чаддаг.
- *KnockoutJS* сангийн файл маш бага бас хөнгөн.
- *KnockoutJS* нь бусад ямар ч бүтцээс хараат бус байдаг. Энэ нь бусад клиент эсвэл сервер талын технологид нийцэх боломжтой.
- *KnockoutJS* -ийн хамгийн чухал нь нээлттэй эх сурвалж бөгөөд ашиглахад үнэ төлбөргүй байдаг.

8.4. KnockoutJS – Програм



Зураг 123. KnockoutJS архитектур

KnockoutJS нь вэб сайтын бүх мэдээллийг нэг хуудсанд ачаалж, динамикаар олж авах боломжийг олгодог. Өөрөөр хэлбэл, домэйн өгөгдөлд *HTML* холбоход маш хялбар болгодог *JavaScript* –ын сан юм. Энэ нь *Model-View View-Model* (*MVVM*) хэмээх загварыг хэрэгжүүлдэг.

`View` нь *HTML* элементүүд болон *CSS* хэлбэрийг ашиглан үүсгэсэн хэрэглэгчийн интерфэйс юм.

```
<body>
    <h2>My Account</h2>
    <span>Bank Name:</span><
    data-bind = "text: account.bank" span id =
"accountBankName"></span>
    <br>
    <span>Account Number</span><
    data-bind = "text: account.accNo" span id =
"accountNum"></span>
    <br>
    <span>Account Type</span><
    data-bind = "text: account.type" span id =
"accountType"></span>
    <br>
    <span>Status</span><
    data-bind = "text: account.status" span id =
"accountStatus"></span>
</body>
```

`View Model` нь өгөгдлийг төлөөлөх шаардлагатай шинж чанар, функцийг агуулсан *JavaScript* объект юм. *View Model* -г өөрчлөхгүйгээр *HTML* -г өөрчлөхөд хялбар болгодог. *KnockoutJS* нь дэлгэц болон өгөгдлийн хооронд шууд холболт үүсгэх боломжийг олгодог.

```
<script type = "text/javascript" src = "jquery-
min.js"></script>
<script type = "text/javascript" src =
"ko/knockout-2.3.0.js">
(document).ready(function(){
    function Account(bank, accNo, type, status){
        var self = this;
        self.bank = bank;
        self.accNo = accNo;
        self.type = type;
        self.status = status;
    };
    function AccountViewModel(){
        var self = this;
        self.account = new Account("golomt",
"11229999", "saving", "active");
    };
    ko.applyBindings(new AccountViewModel());
});
</script>
```

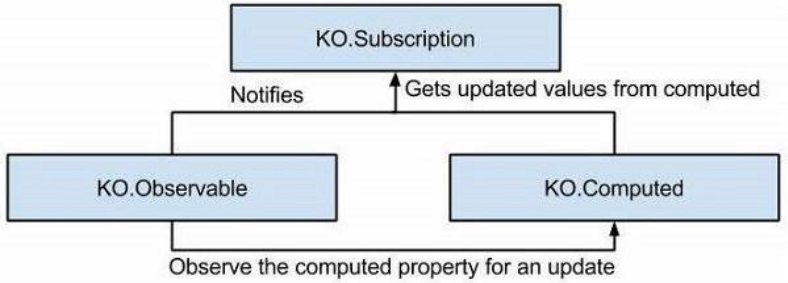
Model бол сервер дээрх домэйн өгөгдөл бөгөөд *View Model* -с хүсэлт илгээх эсвэл хүлээн авсан үед ажиллах боломжтой. Ихэнх тохиолдолд хадгалагдсан өгөгдлийг *Ajax* -аар дамжуулан дуудагддаг.

8.5. Функц үйлдлүүд, хэрэглэх тухай

KnockoutJS нь үндсэн гурван функц дээр тулгуурласан бөгөөд үүнд *observables ба dependency tracking, declarative bindings, templating* багтана.

8.5.1 Observables ба dependency tracking

KnockoutJS -н утгууд шинэчлэгдэх үед автоматаар дагаж өөрчлөгддөг. Энэ үйлдлийг *dependency tracker (ko.dependencyDetection)* гэж нэрлэгддэг нэг объекттой бөгөөд хоёр талын хоорондын завсрын хамаарлаар үүснэ.



Зураг 124. Dependency tracker

Dependency tracker -тэй ажиллах жишээг одоо үзүүлье.

```
<!DOCTYPE html>
<html>
<head>
<title>KnockoutJS How Dependency Tracking
Works</title>
<!-- CDN's-->
<script src =
https://ajax.aspnetcdn.com/ajax/knockout/knockout-
3.1.0.js type = "text/javascript"></script>
</head>

<body>
<div>
    <form data-bind = "submit: addFruits">
    <b>Add Fruits:</b>
    <input data-bind = 'value: fruitToAdd,
valueUpdate: "afterkeydown"'/>
    <button type = "submit" data-bind = "enable:
fruitToAdd().length > 0">Add</button>
    <p><b>Your fruits list:</b></p>
    <select multiple = "multiple" width = "50"
data-bind = "options: fruits"> </select>
    </form>
</div>
```

```
<script>
    var Addfruit = function(fruits) {
    this.fruits = ko.observableArray(fruits);
    this.fruitToAdd = ko.observable("");

    this.addFruits = function() {

        if (this.fruitToAdd() != "") {
            this.fruits.push(this.fruitToAdd());
//Жимс нэмэх
            this.fruitToAdd("");
//Текст боксыг хоослох
        }

    }.bind(this);
// "this" гэдэг нь view модел
    };

    ko.applyBindings(new Addfruit(["Apple",
"Orange", "Banana"]));
</script>
</body>
</html>
```

Дараах зурагт дээрх кодны үр дүнг харуулав.

Зураг 125. Dependency tracker ашигласны үр дүн

8.5.2 Declarative Binding

KnockoutJS -н бас нэг давуу тал нь *UI (User Interface)* рүү өгөгдлийг холбох арга юм. *Bindings* болон *Observables* хоёрыг хооронд хослуулах нь чухал бөгөөд *JavaScript* объектыг *View Model* хэрэглэж болох ба *KnockoutJS* нь *View* -ийг зөв холбох боломжтой. *UI* нь өгөгдлийн шинэчлэлтийг автоматаар шинэчлэж чадахгүй харин *bindings* -ийн тусламжтай шинэчилэгддэг. *Binding* холбоос нь 2 зүйлээс бүрдэх бөгөөд заавал нэр ба утга байна. Дараах жишээг харна уу.

```
Today is: <span data-bind = "text: whatDay>
</span>
```

Энд текст нь заавал байх ёстой нэр, *whatDay* нь заавал байх утга юм.

```
Your name: <input data-bind = "value: yourName,
valueUpdate: 'afterkeydown'"/>
```

Түлхүүр дарагдсаны дараа утгыг шинэчилнэ.

Binding Values (утга): JavaScript -д binding илэрхийлэхийн тулд зөвшөөрөгдсөн утга *single value, literal, a variable* байж болно. Хэрвээ холболт нь зарим хүчингүй илэрхийлэл эсвэл лавлагаа гэсэн утгатай бол *KO* нь алдаа гаргаж, холболтыг боловсруулахаа зогсооно. Жишээ нь:

```
<!-- simple text binding -->
<p>Enter employee name: <input   -bind = 'value:
empName' /></p>
<!-- click binding, call a specific function -->
<button data-bind="click: sortEmpArray">Sort
Array</button>
<!-- options binding -->
<select multiple = "true" size = "8" data-bind =
"options: empArray ,
   selectedOptions: chosenItem"> </select>
```

Дараах зүйлсийг анхаарна уу: *Whitespaces* ямар ч ялгаа байхгүй.

KO 3.0-с эхлээд заавал тодорхойлогдоогүй утгыг өгч болох холболтыг алгасаж болно.

Binding Context (нөхцөл байдал): Одоогийн холболтод хэрэглэгдэж байгаа өгөгдлийг объектоор тайлбарлаж болно. Энэ объектыг холбох нөхцөл гэж нэрлэдэг. *KnockoutJS* – ээр агуулгын шатлалыг автоматаар үүсгэж удирдана. Дараах хүснэгтэд *KO* -н хангаж буй төрөл бүрийн хамаарлын төрлийг жагсаав.

8.5.3 Templating

Template нь *DOM* элементүүдийн багц бөгөөд давтагдах боломжтой. *Templating* нь *DOM* элементүүдийн давхардлыг багасгах өмчийн улмаас цогц програмуудыг бүтээхэд хялбар болгодог. Загвар үүсгэх 2 арга бий.

- Төрөлх үйл явц - Энэ арга нь урагшлах, эсвэл хэрэв байгаа бол хяналтын урсгалын холболтыг дэмждэг. Эдгээр холбоосууд нь элемент дэх *HTML* тэмдэглэгээг авч, үүнийг санамсаргүй зүйлүүдэд загвар болгон ашигладаг. Энэ загварт зориулж гаднах номын сан шаардлагагүй.
- *String* дээр суурилсан *templating* - *KO* нь *ViewModel* утгуудыг дамжуулахын тулд гуравдагч талын хөдөлгүүртэй холбож үр дүнг тэмдэглэхэд документэд оруулдаг. Жишээ нь: *JQuery.tmpl* ба *Underscore Engine*.

Параметрүүд

Дараах шинж чанаруудын хослолыг загварт параметр-утга болгон илгээнэ.

- *name* - энэ нь загварыг харуулж байна.
- *nodes* - Энэ нь загвар болгон ашиглагдах *DOM* зангилааны массивыг илэрхийлнэ. Нэрийн параметрийг дамжуулсан бол энэ параметрийг хэрэгсэхгүй болно.

- *data* - Энэ бол загвараар дамжуулан үзүүлэх өгөгдөл юм.
- *if* - тухайн нөхцөл байдал нь үнэн, үнэн бодитой утга өгвөл загварыг хангана.
- *foreach* - Загварын загварт зориулж загварчлах.
- as зөвхөн *foreach* элемент дэх *alias* үүсгэх явдал юм.
- *afterAdd, afterRender, beforeRemove* - Эдгээр нь гүйцэтгэгдэж буй үйл ажиллагаанаас шалтгаалан гүйцэтгэх боломжтой дуудлагуудыг илэрхийлж байна.

Binding Context төрөл ба майлбар [5]

Root: Энэ нь үргэлж *ViewModel* дээд түвшинд хамаарна. Энэ нь *ViewModel* -г ашиглах дээд түвшний аргуудад хандах боломжийг бүрдүүлдэг. Ихэвчлэн *o.applyBindings* руу дамжуулагдах объект юм.

Data: JavaScript объект дээр энэ түлхүүр үгстэй маш их төстэй. Холбогдлын үл хамаарах зүйл нь одоогийн нөхцөл дэх *ViewModel* объектыг хэлнэ.

Index: Энэ нь *foreach* давталтын дотор массивын одоогийн элементийн индексийг агуулна. Массив шинэчлэгдэж байх үед индексийн утга автоматаар өөрчлөгддөг.

Parent: Энэ *property* нь *ViewModel* объектыг хэлнэ. Энэ нь давталт доторхи *ViewModel* шинж чанараас гадуур нэвтрэхийг хүсч байгаа үед ашигтай байдаг.

Index: Удамшлын түвшинд хамаарах объектийг *parentContext* гэж нэрлэдэг. Энэ нь эцгээс ялгаатай. Харин *parentContext* гэдэг нь холбогдох нөхцөлийг хэлнэ. Тухайлбал, гадна талын индексд дотоод нөхцөл байдлаас хандах хэрэгтэй байж магадгүй.

Rawdata: Энэ нөхцөл байдал нь өнөөгийн нөхцөл байдалд *ViewModel* утгыг агуулдаг. Энэ нь өгөгдөлтэй төстэй. *ViewModel* болон *rawdata* нь бодит ажиглалттай мэдээлэл

болдог.

Component: Энэ нь тухайн бүрэлдэхүүн хэсгийн дотор байх үед тухайн бүрэлдэхүүн хэсгийн *ViewModel* -г лавлана гэсэн үг юм. Тухайлбал, бүрэлдэхүүн хэсгийн загварыг харуулахын оронд *ViewModel* -с зарим эд хөрөнгөд нэвтрэхийг хүсч болно.

ComponentTemplateNodes: Энэ нь тусгай бүрэлдэхүүн хэсгийн загвар дотор байхад тухайн бүрэлдэхүүн хэсэг рүү дамжигдах *DOM* зангилааны массивыг илэрхийлнэ.

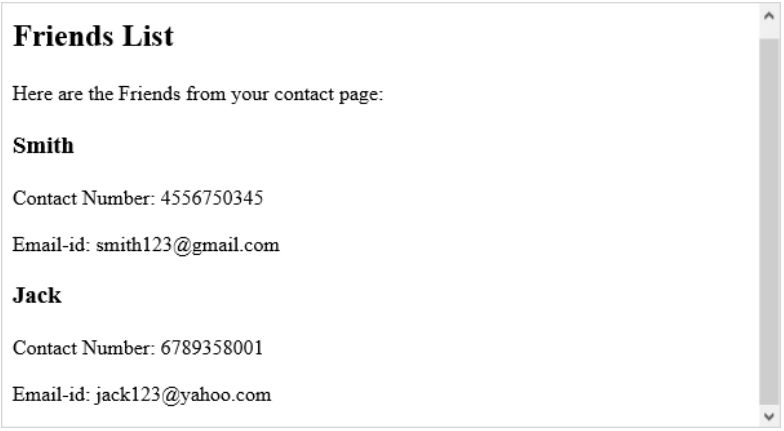
Binding Context төрлийн тайлбар: Загварууд нь хяналтын урсгалын холболтод хэрэглэгдэх үед *DOM* доторх *HTML* тэмдэглэгээгээр далдлагдсан байдаг. Хэрвээ та хүсвэл маягтыг тусдаа элемент болгож, дараа нь нэрээр нь нэрлэж болно.

```
<!DOCTYPE html>
<head>
<title>KnockoutJS Templating - Named
Template</title>
<script src =
"https://ajax.aspnetcdn.com/ajax/knockout/knockout
-3.3.0.js"
    type = "text/javascript"></script>
</head>
<body>
<h2>Friends List</h2>
Here are the Friends from your contact page:
<div data-bind = "template: { name: 'friend-
template', data: friend1 }"></div>
<div data-bind = "template: { name: 'friend-
template', data: friend2 }"></div>
<script type = "text/html" id = "friend-template">
    <h3 data-bind = "text: name"></h3>
    <p>Contact Number: <span data-bind = "text:
contactNumber"></span></p>
    <p>Email-id: <span data-bind = "text:
email"></span></p>
</script>
```

```
<script type = "text/javascript">
    function MyViewModel() {
    this.friend1 = {
        name: 'Smith',
        contactNumber: 4556750345,
        email: 'smith123@gmail.com'
    };
    this.friend2 = {
        name: 'Jack',
        contactNumber: 6789358001,
        email: 'jack123@yahoo.com'
    };
    }
    var vm = new MyViewModel();
    ko.applyBindings(vm);
</script>
</body>
</html>
```

Зураг 126. template-named.html файл

Дээрх жишээний үр дүнг дараах зурагт харуулав.



Зураг 127. template-named.html үр дүн

8.6. Гадаад холболтууд

8.6.1 Visible холболт

DOM элементийг холбоход ашиглагдах бөгөөд далд эсвэл харагдах байдлаар тохируулж болно [6]. Тухайлбал:

```
<div data-bind="visible: shouldShowMessage">
    "shouldShowMessage" зөвхөн true утгатай болох
үед л харах болно.
</div>

<script type="text/javascript">
    var viewModel = {
        shouldShowMessage: ko.observable(true)
        //эхлээд зурвас харуулах
    };
    viewModel.shouldShowMessage(false);
    // ... зурвасыг нуусан
    viewModel.shouldShowMessage(true);
    // ... зурвасыг харуулах
</script>
```

Зураг 128. "visible" холболт жишээ 1

Параметрүүд

Дээрх жишээнний параметрт *false* утгыг тодорхойлсон байна. *yourElement.style.display* байгаа үед таны холбоосыг *yourElement.style.display none* гэж үзнэ. Энэ нь *CSS* -р тодорхойлогдсон ямар ч дэлгэцийн хэв маягт илүү чухал байдаг.

Дараа нь параметрт *true* утгад тогтоогдвол *binding yourElement.style.display* утгыг устгаж, түүнийг харагдахаар болгоно. Өөрийн *CSS* дүрэмд тодорхойлсон загварыг ашиглана гэдгийг анхаарна.

Хэрэв параметр утгатай бол утга өөрчлөгдөх бүрт элемент шинэчлэгдэх болно. Хэрэв параметр ажиглагдахгүй бол

зөвхөн элементийн харагдах байдлыг нэг удаа тодорхойлоод, дараа нь дахин шинэчлэхгүй.

Элементийн *visible* -г хянахын тулд функц болон илэрхийллийг ашиглах *JavaScript* функц эсвэл дурын *JavaScript* илэрхийллийг ашиглаж болно. Хэрэв *KO* функцийг ашиглан элементийг нуух эсэхийг тодорхойлохын тулд доорх кодыг ашиглаж болно.

```
<div data-bind="visible: myValues().length > 0">
    'myValues' дор хаяж нэг утгатай болвол үр дүнг
харна.
</div>

<script type="text/javascript">
    var viewModel = {
        myValues: ko.observableArray([])
        //Эхлэх үед ямар ч утгагүй тул үр дүн
нуугдсан байна.
    };
    viewModel.myValues.push("some value");
    // Одоо үр дүн харагдана.
</script>
```

Зураг 129. Visible холболт жишээ 2

8.6.2 Текст холболт

Холбогдох текст нь хамааралтай *DOM* элементийн параметр дахь текстийн утгыг харуулдаг. Өөрөөр хэлбэл, энэ нь уламжлалтаар текст харуулдаг **** эсвэл **** шиг элементүүдэд ашигтай боловч техникийн хувьд үүнийг аль ч элементээр ашиглаж болно. Тухайлбал:

```
Today's message is: <span data-bind="text:
myMessage"></span>

<script type="text/javascript">
```

```
    var viewModel = {
        myMessage: ko.observable()
        //Анхандаа хоосон    };
    viewModel.myMessage("Hello, world!");
    // Текст үзэгдэнэ
</script>
```

Зураг 130. Text холболт

Параметрүүд

Knockout нь элементийн агуулгыг тухайн параметрийн утгатай текст зангилаа руу заана. Өмнөх агуулгыг дарж бичиж болно.

Хэрэв тухайн параметр нь ажиглагдах утгатай бол энэ утга өөрчлөгдөх бүрт элементийн текстийг шинэчлэх болно. Хэрэв параметрийн утга ажиглагдахгүй бол энэ нь зөвхөн элементийн текстийг нэг удаа тохируулах бөгөөд дараа дахин шинэчилнэ.

Хэрэв объект эсвэл массив зэрэг өөр хэлбэрийн зүйл дамжуулбал дэлгэцийн текст yourParameter.toString() хөрвүүлэлт хийж үр дүнг гаргана.

Текстийн утгыг илэрхийлэхийн тулд функц ашиглаж болох бөгөөд үүнийг дараах жишээнээс харна уу.

```
The item is <span data-bind="text:
priceRating"></span> today.
<script type="text/javascript">
    var viewModel = {
        price: ko.observable(24.95)    };
    viewModel.priceRating =
ko.pureComputed(function() {
        return this.price() > 50 ? "expensive" :
"affordable";
    }, viewModel);
</script>
```

Зураг 131. Text холболтод функц ашиглах

Дээрх жишээний текстийн үнэ өөрчлөгдөх бүрт “үнэтэй”, “хямд” хооронд шилжих боллоо. Өөрөөр хэлбэл, иймэрхүү энгийн зүйлийг хийж байгаа бол тооцоолж болохуйц тооцооллыг үүсгэх шаардлагагүй болно. Текстийг заавал дурын *JavaScript* илэрхийлж болно. Жишээлбэл, **`The item is <span data-bind="text: price() > 50 ? 'expensive': 'affordable'></span> today.`** Үнэ нь яг ижил үр дүнтэй байна.

8.6.3 HTML холболт

HTML заагч нь параметрийн тодорхойлсон *HTML DOM* элементийг үүсгэдэг.

```
<div data-bind="html: details"></div>

<script type="text/javascript">
    var viewModel = {
        details: ko.observable()
    };
    viewModel.details("<em>For further details,
view the report <a
href='report.html'>here</a>.</em>");
    // HTML агуулга харагдана
</script>
```

Зураг 132. Html холболтын жишээ

Параметрүүд

KO нь өмнөх агуулгыг арилгаж, *jQuery* -н *html* функцийг ашиглан параметрийн утгыг элементийн агуулгаар тодорхойлдог.

Хэрэв тухайн параметр ажиглагдах утгатай бол энэ утга өөрчлөгдөх үед элементийн агуулгыг шинэчлэх болно. Хэрэв параметр ажиглагдахгүй бол зөвхөн элементийн агуулгыг нэг

удаа тохируулах бөгөөд дахин шинэчилнэ.

8.6.4 CSS холболт

CSS холболт нь нэмэх, устгах эсвэл *CSS* нэртэй нэгдэж байгаа классуудыг холбогдох *DOM* элемент рүү шилжүүлнэ. Жишээлбэл, тухайн утга сөрөг байвал тэдгээрийг улаан өнгөөр харуулах байж болно.

```
<div data-bind="css: { profitWarning:
currentProfit() < 0 }">
   Profit Information
</div>

<script type="text/javascript">
    var viewModel = {
        currentProfit: ko.observable(150000)
        //Эерэг утга
    };
    viewModel.currentProfit(-50);
    //Одоо "profitWarning" class тодорхойлогдоно
</script>
```

Зураг 133. CSS холболт статаик классын жишээ

```
<div data-bind="css: profitStatus">
   Profit Information
</div>

<script type="text/javascript">
    var viewModel = {
        currentProfit: ko.observable(150000)
    };
    // Эерэг утга учир "profitPositive" классыг
тодорхойлно
    viewModel.profitStatus =
ko.pureComputed(function() {
```

```
        return this.currentProfit() < 0 ?
"profitWarning" : "profitPositive";
    }, viewModel);

    // "profitPositive" класс устгагдсан учир
"profitWarning" класс нэмэгдэнэ
    viewModel.currentProfit(-50);
</script>
```

Зураг 134. CSS холболт динамик классын жишээ

Параметрүүд

Хэрэв та статик *CSS* классын нэрийг ашиглаж байгаа бол, та *CSS* ангиллын нэрс нэрс, утгыг *JavaScript* -д дамжуулж болно.

Олон CSS классуудыг нэг дор тохируулж болно. Мөн ижил нөхцөл дээр тулгуурлан олон *CSS* ангиллыг тохируулж болно.

Хэрэв параметр ажиглагдах утгыг зааж өгвөл ажиглагдах утга өөрчлөгдөх бүрт *CSS* класс нэмэх эсвэл хасах болно. Хэрэв параметр ажиглагдах утгыг зааж өгөөгүй бол зөвхөн класс нэмэх буюу хасна.

Ердийн үед та дурын *JavaScript* илэрхийлэл эсвэл функцийг параметрийн утга болгон ашиглаж болно. *KO* нь тэдгээрийг үнэлэх ба үр дүнг ашиглахын тулд тохирох *CSS* классуудыг нэмэх эсвэл хасах боломжтой болно.

8.6.5 Style холболт

Загварын холбогдлыг нэмсэн эсвэл нэг буюу хэд хэдэн загварын утгыг холбогдох *DOM* элемент рүү шилжүүлдэг. Дараах жишээнд сөрөг утгатай байвал зарим утгыг тодруулах, эсвэл өөрчлөгдсөн тоон утгатай тааруулахын тулд **`bar`** -ийн өргөнийг тохируулах хэрэгтэй.

```
<div data-bind="style: { color: currentProfit() <
0 ? 'red' : 'black' }">
   Profit Information
</div>

<script type="text/javascript">
    var viewModel = {
        currentProfit: ko.observable(150000)
        // эерэг утга тул хар байна
    };
    viewModel.currentProfit(-50);
    // Одоо тухайн DIV агуулга улаан болно
</script>
```

Зураг 135. Style холболтын жишээ

8.6.6 Attr холболт

Attr холболт нь холбогдох *DOM* элементэд ямар нэг шинж чанарын утгыг тохируулах еренхий арга юм. Жишээ нь, элементийн гарчиг, *img* -н *src*, эсвэл *href* холбоосыг автоматаар шинэчилж байх зэрэг асуудлуудыг шийдэж болно.

```
<a data-bind="attr: { href: url, title: details
}">
    Report
</a>

<script type="text/javascript">
    var viewModel = {
        url: ko.observable("year-end.html"),
        details: ko.observable("Report including
final year-end statistics")
    };
</script>
```

Зураг 136. Attr холболтын жишээ

Энэ нь элементийн *href* атрибутыг жилийн эцсийн буюу *year-end.html* хүртэлх элементийн нэрийг тайланд багтаасан эцсийн жилийн статистик болгож тохируулах болно.

8.7. Удирдлагын холболтууд

8.7.1 Foreach холболт

Массив, олонлог дотроос утгуудыг давтаж харах шаардлагатай бол энэ холболтыг ашиглана. Одоо массив дээр энэ давталтын хэлбэрийг ашиглах жишээг харна уу.

```
<table>
<thead>
    <tr><th>First name</th><th>Last name</th></tr>
</thead>
<tbody data-bind="foreach: people">
    <tr>
        <td data-bind="text: firstName"></td>
        <td data-bind="text: lastName"></td>
    </tr>
</tbody>
</table>

<script type="text/javascript">
ko.applyBindings({
    people: [
{ firstName: 'Bert', lastName: 'Bertington' },
{ firstName:'Charles', lastName: 'Charlesforth' },
{ firstName: 'Denise', lastName: 'Dentiste' }
 ]
});
</script>
```

Зураг 137. Foreach ашиглах нь

Одоо *foreach* давталтыг нэмэх, устгахтай хамтран ашиглаж үзье.

```
<h4>People</h4>
<ul data-bind="foreach: people">
    <li>
        Name at position <span data-bind="text:
$index"> </span>:
        <span data-bind="text: name"> </span>
        <a href="#" data-bind="click:
$parent.removePerson">Remove</a>
    </li>
</ul>
<button data-bind="click: addPerson">Add</button>
```

Зураг 138. View хэсэг

```
function AppViewModel() {
    var self = this;

    self.people = ko.observableArray([
        { name: 'Bert' },
        { name: 'Charles' },
        { name: 'Denise' }
    ]);
    self.addPerson = function() {
        self.people.push({ name: "New at " + new
Date() });
    };

    self.removePerson = function() {
        self.people.remove(this);
    }
}
ko.applyBindings(new AppViewModel());
```

Зураг 139. View Model хэсэг

Дараах зурагт үр дүнг харуулж байна.

Зураг 140. foreach давталт ашигласан жишээний үр дүн

8.7.2 if холболт

Хэрэв тодорхой нөхцөлөөс шалтгаалж үр дүнг гаргах шаардлагатай бол энэ холболтыг ашиглана. Дараах жишээнд холбогдох утгууд өөрчлөгдвөл динамикаар нэмэх, устгах нөхцөлийг харуулж байна.

```
<label><input type="checkbox" data-bind="checked:
displayMessage" /> Display message</label>

<div data-bind="if: displayMessage">Here is a
message. Astonishing.</div>
```

Зураг 141. Html файлд If ашиглах

```
ko.applyBindings({
    displayMessage: ko.observable(false)
});
```

Зураг 142. View Model хэсэг

8.7.3 Component холболт

Холбох бүрэлдэхүүн хэсэг нь тодорхой бүрэлдэхүүн хэсгийг элементэд оруулж, нэмэлт сонголтуудыг дамжуулдаг.

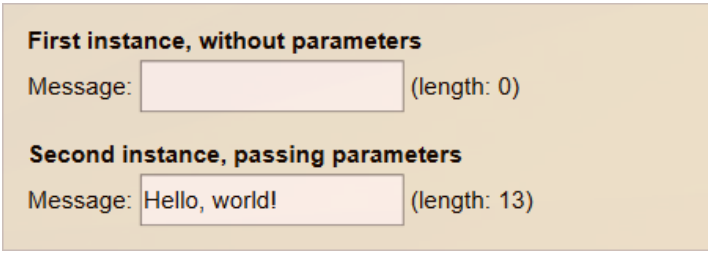
```
<h4>First instance, without parameters</h4>
<div data-bind='component: "message-
editor"'></div>

<h4>Second instance, passing parameters</h4>
<div data-bind='component: {
    name: "message-editor",
    params: { initialText: "Hello, world!" }
}'></div>
```

Зураг 143. Html файлд компонентыг ашиглах нь

```
ko.components.register('message-editor', {
    viewModel: function(params) {
        this.text = ko.observable(params &&
params.initialText || '');
    },
    template: 'Message: <input data-bind="value:
text" /> '
            + '(length: <span data-bind="text:
text().length"></span>)'
});

ko.applyBindings();
```

Зураг 144. View Model хэсэгт компонентыг дүрслэх нь

Дараах зурагт дээрх жишээний үр дүнг харуулж байна.



Зураг 145. Үр дүн

Ашигласан материал

[1] MZM.TEAM, "MZM agency," [Online]. http://www.mzm.agency/

[2] "Meteor," Meteor Development Group Inc, [Online]. https://www.meteor.com/.

[3] "Angular Dart," [Online]. https://webdev.dartlang.org/angular/guide/architecture.

[4] "BackboneJS," [Online]. http://backbonejs.org/.

[5] "KnockoutJS Tutorial," Tutorial point, [Online]. https://www.tutorialspoint.com/knockoutjs/knockoutjs_declarative_bindings.htm.

[6] "Knockout," [Online]. http://knockoutjs.com/documentation/visible-binding.html.