eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Gradu Amaierako Lana

Informatika Ingeniaritzako Gradua

Software Ingeniaritza

Elkarbackup software librearen automatizaziorako RESTful APIaren garapena

Uxue Anabitarte Soroa

Zuzendariak
Eneko Lacunza
Rosa Arruabarrena

2021eko maiatzaren 25

Esker onak

Hasi aurretik eskerrak eman nahiko nizkieke Gradu Amaierako Lan hau aurrera eramaten lagundu didatenei. Asko ikasi dut proiektu hau garatzen, eta ziur nago etorkizunerako jasotakok esperientzia oso baliagarria izango zaidala.

Rosa Arruabarrenari, Gradu Amaierako Lan honen tutore izateko prest egon eta proiektua aurrera eramaten laguntzeagatik.

Eneko Lacunzari, *Binovo* enpresako tutoreari, enpresan praktiketan aritzeko aukera emateagatik eta momentu oro laguntzeko prest azaltzeagatik.

Igor Blancori, proiektuaren egunerokotasunean zalantza guztiak pazientziaz argitu eta laguntzeagatik.

Laburpena

Gradu Amaierako Lan honetan *Binovo* enpresaren *Elkarbackup* aplikazioaren RESTful API bat garatuko da, horrekin batera API-a probatzeko testak eta API-a kontsumitzeko komando-lerroko bezeroa inplementatuz.

Binovo softwarearen garapen eta mantenuaz arduratzen den enpresa bat da, eta bere produktuetako batzuk bezeroei ezartzean *Elkarbackup* aplikazioa ere instalatzen zaie datuen segurtasun-kopiak kudeatu eta egikaritzeko. Aplikazio honen konfigurazio-prozesua automatizatzeko garatuko da RESTful API-a.

Proiektu hau Euskal Herriko Unibertsitateko Informatika Ingeniaritza Graduko ikasle batek burutua izango da *Binovo* enpresarentzat.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1

Sarrera

Gradu Amaierako Lan honi hasiera eman eta testuinguruan jartzeko, atal honetan proiektu hau proposatu duen enpresa aurkeztu, bertan aurretik izandako esperientzia azaldu eta proiektu honetan landuko den aplikazioaren ikuspegi orokor bat emango da.

1.1 *Binovo* enpresa

Proiektu hau *Binovo* enpresan garatu da. *Binovo*[12] Oiartzuneko (Gipuzkoa) software garapen eta mantentzean lan egiten duen enpresa bat da. IT arloko soluzio berritzaileak garatu eta inplementatzen ditu *Agile* korronteetan oinarritutako metodologien bidez bezeroen beharrei erantzutearren.

Oraintxe bertan, lan-arlo nagusiak dira enpresen kudeaketarako ERP-ak, zereginen automatizaziorako RPA (prozesuen automatizazio robotikoa), sistemen kudeaketa eta neurrira egindako software garapena.

1.2 Praktikak *Binovon*

Egileak Gradu Amaierako Lana *Binovo* enpresan garatuko zuela jakinda, aurretik praktikak egiteko aukera planteatu zion enpresako tutoreari. Praktika hauek enpresa eta proiektuan landuko den *Elkarbackup* aplikazioa ezagutu eta lantzeko aukera eman diote egileari. Izan ere, aplikazioaren lan-ingurunearen *upgrade*-az arduratu da, horrela Gradu Amaierako Lanerako ikasketa-prozesua aurretik hasiz.

1.3 *Elkarbackup* aplikazioa

Elkarbackup aplikazioa segurtasun-kopiak modu erosoan egin eta kudeatzeko software bat da. *Tknika*[1]zentroarentzat garatua izan zen, Eusko Jaurlaritzaren Hezkuntza Sailaren Lanbide Heziketako Sailburuordetzak bultzatutako zentroarentzat hain zuzen. Hala ere, duen potentziala ikusita, Binovok hobekuntzak egiten jarraitu eta berak ere erabiltzen du bai bezeroetan, baita beraien sisteman ere.

[1] www.tknika.eus

2

Proiektuaren kudeaketa plana

Memoriaren atal honetan proiektua arrakastaz burutzeko zehaztutako plana aurkeztuko da. Horretarako, helburuak eta irismenaren nondik norakoak azaldu eta hauek betetzeko hasiera batean ezarritako plangintzarekin amaituko da.

2.1 Aurrekariak

Sarreran (1 atalean) aipatu bezala, Gradu Amaierako Lan hau *Binovo* enpresaren beharretatik sortutako proiektu bat da. Gaur egun, *Binovo*ren lan eremu nagusietako bat da ERP[1]-en garapena *Odoo*[2] plataformarekin. Eusko Jaurlaritzak 2022rako ezarriko duen legediaren arabera[14], jarduera ekonomiko bat duen orok, bai enpresek bai autonomoek, fakturazio-software bat izan beharko dute *TicketBAI*[3] sistemarekin txertatuta, faktura guztiak Ogasunera bidali ahal izateko. *Binovo* hasia da dagoeneko inplementazio hori eskaintzen, eta horregatik, bezero berri askoren sarrera aurreikusten da, eta bezero horietako bakoitzak, *Elkarbackup* izango du segurtasun-kopiak kudeatzeko. Hori horrela izanda, *Binovo*k bezeroen instalazio eta konfiguraziorako *Ansible*[4] automatizazio tresna erabili nahi du prozesua automatizatzeko. Helburu honen baitan, *Elkarbackup* aplikazioaren web-bezeroen konfigurazioa automatizatzeko aukera eduki nahi da, eta hori izango litzateke proiektu honen helburua: *Elkarbackup* bezeroen konfigurazio-prozesua automatizatzeko aukera izateko *RESTful API* interfaze baten inplementazioa.

2.2 Helburua

Aipatu bezala, proiektu honen helburu nagusia *Elkarbackup* aplikazioaren *REST API* bat garatzea izango da; *Binovo* enpresak duen *Elkarbackup*-en konfigurazio-prozesuaren automatizazioaren xedearen baitan. API honek hainbat oinarrizko funtzionalitate izan beharko ditu, eta behin oinarrizko beharrak asetuta, ahalik eta API osatuena inplementatzea izango da helburua. Hauek izango lirateke helburu zehatzak:

[1] *ERP* edo *Enpresa Baliabideen Plangintza*, negozioen esparru ezberdinen kudeaketa ahalbidetzen duten softwareak dira. Hala nola, ekoizpena, finantzak eta giza-baliabideak.

[2] Odoo, https://www.odoo.com/

[3] TicketBAI, www.ticketbai.eus

[4] Ansible, https://www.ansible.com/

- Plangintza zuzen eta errealista bat osatzea, ahalik eta orduen estimazio zehatzena osatuz, honen desbiderapena txikia izan dadin.
- Jarraipen eta kontrol eraginkor bat aurrera eramatea. Proiektuaren emaitzak esperotakoa bete dezan, zehaztutako epe eta baldintzen barruan.
- Proiektuaren helburu nagusiak betetzea, hau da, *Elkarbackup* programarentzat API bat garatzea, enpresak ezarritako funtzionalitate eta testak aurrera eramanez, zehaztutako baldintzekin.

2.3 Irismena

Gradu Amaierako Lan honen irismenak puntu hauek bilduko diru:

1. API bat garatu beharko da *Elkarbackup* aplikazioarentzat Gradu Amaierako Lan honetan, REST software arkitektura estiloa jarraituz eta aplikazioaren oinarrizko funtzionalitateak inplementatuz.

   Funtzionalitate hauek izango dira, ordena mantenduz:

   a) Bezeroak sortu, zerrendatu, editatu eta ezabatu.

   b) Lanak sortu, zerrendatu, editatu eta ezabatu.

   c) Funtzionalitate hauek burutu ahal izateko autentifikazioa.

2. Test automatikoen inplementazioa garatutako API-aren funtzionamendua frogatzeko eta etorkizun batean ere aldaketarik balego, modu erraz batean egiaztapenak egin ahal izateko.

3. Komando-lerrotik API-a atzitzeko web-bezeroaren garapena.

Aipatutako ataza horiek egoki burutzen badira, esan daiteke proiektuak arrakasta izan duela. Dena den, plangintzaren arabera eta lan-erritmoaren arabera, baliteke funtzionalitate gehigarriren bat sartzeko aukera izatea. Esaterako, *scriptak* zerrendatu, sortu eta ezabatzeko aukera.

2.4 Plangintza

Atal honetan proiektuaren plangintza jasotzen da. Horretarako, egin beharrekoa lan-paketeetan banatu da lehenik 2.1 irudian, ondoren pakete bakoitzeko atazak zehaztu eta ataza bakoitzari denbora estimazio bat egin (2.1 taulan) eta bakoitzaren epeak aurreikusteko.

2.4.1 Lanaren Deskonposaketa Egitura (LDE)

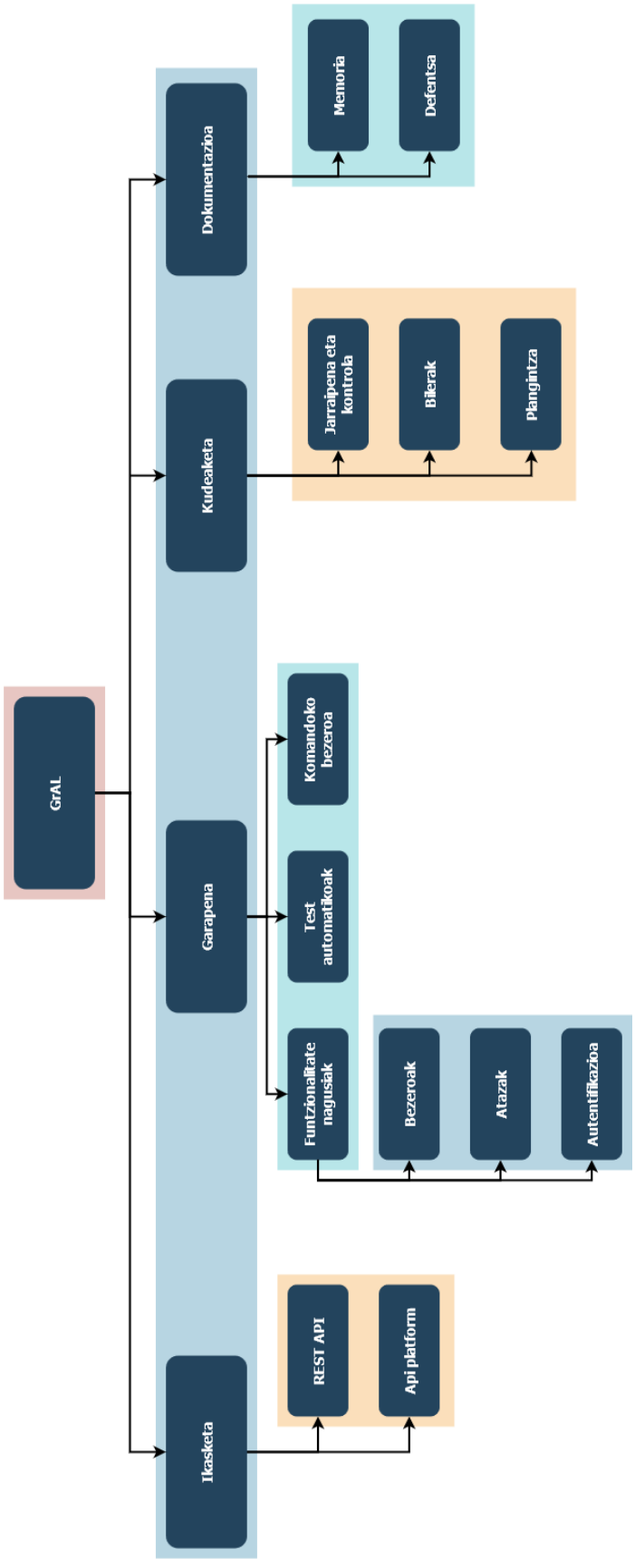
Proiektuaren deskonposaketa LDE diagrama baten bitartez azaldu da, beherako ikusi daitekeen bezala. Lau atal nagusi bereizi dira lan honetan: Ikasketa, Garapena, Kudeaketa eta Dokumentazioa.

Ikasketan bi atal nabarmendu dira: API REST kontzeptuaren inguruko formakuntza eta API Platform tresnaren erabileraren ikasketa; hau da, API-a bera nola garatuko den eta honek eskaintzen dituen aukera anitzak ulertu eta interesatzen direnak barneratzeko.

Garapenean hiru atal bereizi daitezke. Batetik, API-ak izango dituen funtzionalitate nagusiak, hau da, bezero eta atazak sortu, ezabatu, zerrendatu eta editatzea eta hori burutu ahal izateko kautotzearen integrazioa. Bestalde, funtzionalitate hauek behar bezala funtzionatzen dutela egiaztatuko duten test automatikoak inplementatu eta beharrezko frogak egin. Eta azkenik, APIa komando lerrotik atzitu ahal izateko bezeroaren inplementazioa.

Kudeaketari dagokionez, atal honen barruan aurkituko lirateke jarraipen eta kontrol prozesua; bai proiektuaren zuzendariarekin, bai enpresako tutorearekin egindako bilerak; eta proiektua martxan jarri eta gidatzeko zehaztutako plangintza.

Azkenik, dokumentazioaren inguruan burutu beharreko atazak izango dira Gradu Amaierako Lanaren memoria eta behin proiektua burututa, honen defentsa.



2.1 Irudia: LDE diagrama

2.4.2 Estimazioak

2.1 taulan ikus daitekeenez, denboraren zati handiena garapenari esleitu zaio. APIaren inplementazioa izango baita lan-karga gehien suposatuko duena, baita honen probak ere (denera 180 ordu).

Horren ondoren, lan-pakete handiena dokumentazioa izango da, horren barruan baitago Gradu Amaierako Lan honen memoria burutzea; eta honen luzeera eta lanketa maila ikusita, ordu kopuru esanguratsu bat dedikatu beharko zaio (90h). Defentsari ordea ez zaio horrenbeste denbora dedikatuko (20h), izan ere, proiektu osoan zehar lortutako jakintzarekin, ataza hau egiteko gaitasunak egongo dira eta ez zaio denbora asko esleitu beharko.

Hurrengo lan-karga kudeaketarena izango da, proiektuaren jarraipen eta kontrola (20h), bilerak (15h) eta plangintza (15h) osatzearen ordu kopurua parekatuta dagoelarik. Kontuan izan behar da proiektuaren plangintza hasieran definituko dela, eta beraz proiektuaren hasieran egongo dela ordu karga hori, eta ondoren jarraipena egin beharko zaiola aurreikuspen horiei proiektuak arrakasta izan dezan.

Azkenik lan-karga txikiena ikasketak izango du. Azken finean, aipatu behar da Gradu Amaierako Lan hau egiten duen ikasleak *Elkarbackup* programaren eta hau garatuta dagoen *Symfony* plataformaren lanketa aurretik burututako praktiketan bereganatu duela, eta beraz ikasketa API-aren kontzeptuaren eta hau garatzeko tresnak ezagutzean zentratuko dela.

Ataza	Estimazioa (h)
Ikasketa	20
Rest API	5
API-platform	15
Garapena	180
Funtzionalitate nagusiak	80
Test automatikoak	50
Komandoko bezeroa	50
Diseinua	10
Kudeaketa	50
Jarraipena eta kontrola	20
Plangintza	15
Bilerak	15
Dokumentazioa	90
Memoria	70
Defentsa	20
Denera	350

2.1 Taula: Atazen ordu-estimazioa

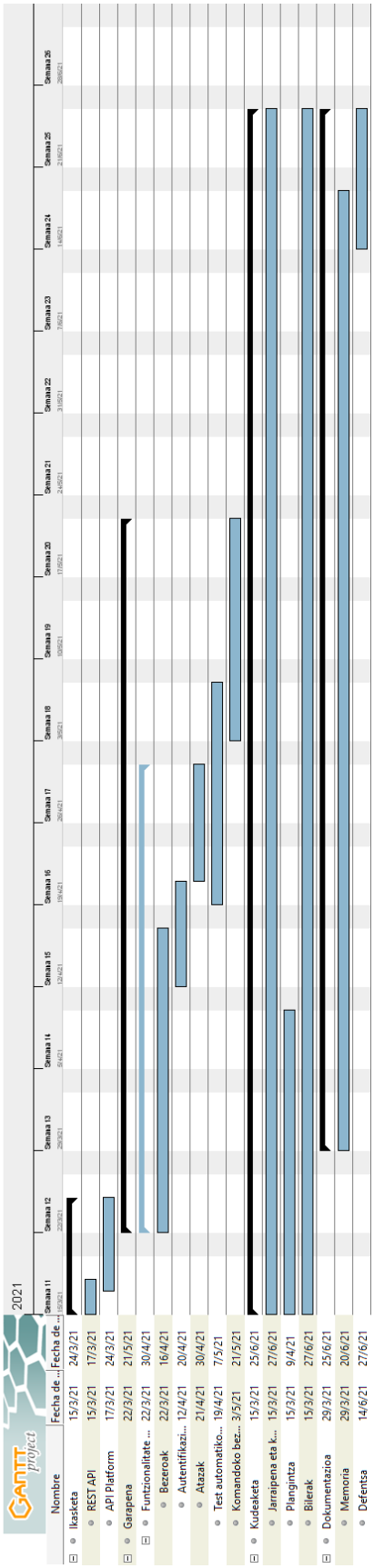
2.4.3 Gantt diagrama

Atazen epeen estimazioa eta proiektuaren garapena modu grafikoan ikus dezakegu 2.2 irudiko Gantt diagramaren bitartez. Proiektuaren plangintza eta oinarrizko ikasketari dedikatuko zaio denbora prozesuaren lehen asteetan. Jarraipena eta kontrola berriz, zentzuzkoa den bezala, proiektu osoan zehar egongo da presente, tutore eta zuzendariekin bilerak bezala. Bi ataza hauekin, proiektuaren egoeraren berri izango da une oro, desbiderapenak, akatsak eta arazoak identifikatuta izateko aukera edukiz.

Garapenari dagokionez, bezeroaren zatiaren garapenari eman zaio eperik luzeena, lehenengoa izanik, erritmoa hartzea gehiago kostatuko delakoan. Ondoren kautotzea garatuko da APIaren amaierako funtzionamendura gerturatzen joateko, eta azkenik atazak zerrendatu, sortu, ezabatu eta editatzearen zatia burutuko da. Behin inplementazio hau eginda, test automatikoak garatuko dira, frogapen guztiak egiteko eta funtzionalitate nagusiei beharrezko aldaketak egiteko. Garapeneko azken ataza izango da komandoko bezeroa garatzea.

Azkenik, dokumentazioaren lan-paketea dugu. Honen barruan, memoria proiektuaren bizi-ziklo osoan garatzen joango den ataza bat izango da. Gradu Amaierako Lanaren defentsa ordea, behin memoria bukatzean hasiko da sortzen eta aurkezpen datarako bukatuta egon

beharko du.



2.2 Irudia: Gantt diagrama

2.5 Arriskuak

- **Informazio galera** izan daiteke kalte larrienetakoak suposatuko lituzkeen arriskuetako bat, denbora estimazioetan eragin handia izango lukeelako eta baliteke proiektua epeen barruan burutzeko estutasunak izatea. Dena den, egia da ez dagoela probabilitate handiegia hau gertatzeko. Azken finean, hau saihesteko neurriak hartuko direlako, une oro materialaren segurtasun-kopiak eginez eta abar.
- **Osasun arazoak** ere beste faktore bat izan daiteke, ohi baina garrantzitsu eta probableagoak, bizi dugun pandemia egoera dela eta. Ez soilik gure ondoezak, baizik eta inguruko norbaitenak eragina izan dezakeelako gure lan erritmoan, konfinatu beharko bagina adibidez. Arrisku honek desbiderapen garrantzitsu bat ekar dezake proiektuaren epeetan.
- **Teknologia arazoak.** Proiektua ikaslearen etxetik burutuko denez, aipatu behar da baliabideak ez direla idealak, eta kontuan izan beharko dira konexio arazo puntualak eta antzeko arazo teknologikoak.
- **Diseinu desegoki** batek eragin handia izango luke Gradu Amaierako Lanaren bizizikloan. Azkenean puntu honetan hartutako erabakiek baldintzatuko dute proiektuaren garapen lana. Zati honetan erabaki okerrak hartzen badira, prozesua korapilatu daiteke.
- **Komunikazio falta.** Enpresa batek proposatutako proiektua izanik, ezinbesteko izango da proiektuak honen beharrak asetzea, eta horretarako komunikazio sendoa oinarrizkoa da. Aipatu bezala, ikasleak etxetik egin beharko duenez proiektua, komunikazio guztiak mezu edo bideodei bidez izango dira enpresako zuzendari eta kideekin. Horrek gaizki-ulertu batera eraman dezake, atazaren bat errepikatu edo berrikustea ekar dezakeena.
- **Estimazio okerra** oso probabilitate altua duen arriskua da. Aurretiko plangintza zuhur eta errealista batekin kalteak murriztu daitezke, eta aldi berean, gakoa izango da proiektuaren jarraipena eta kontrola burutzea, edozein desbiderapen identifikatu eta plangintzan beharrezko doikuntzak egiteko.

2.6 Kalitate-plana

Amaierako produktuak esperotako kalitate maila duela ziurtatzeko, honen hainbat aspektu hartu beharko dira kontuan. Ezarritako baldintza hauekin, enpresaren beharrak asetzea espero da.

- **Kodearen kalitateari** dagokionez, honek egitura ulergarri eta koherentea izan beharko du. Betiere enpresak aurrez ezarritako estandarra jarraituz, etorkizunean egilea ez den beste norbaitek kodea eskuratzen badu, erraz ulertu eta nahi dituen aldaketak egiteko gai izan dadin.
- **Dokumentazioaren kalitatea.** Bai memorian bai sor daitezkeen beste dokumentuetan, hauen formatu, estilo eta edukia zainduko da. Beharrezkoa dena modu xehe batean azalduz, hizkuntza jasoan betiere eta formatu txukun batean.

- **Funtzionalitateen kalitatearen** garrantzia ia aipatu beharrik ere ez dago. Amaierako produktuan inplementatutako funtzionalitateek modu zuzen batean bete beharko dituzte hasieran zehaztutako eskakizunak. Horretarako ezinbestekoa izango da probak modu sistematiko eta zuhurrean burutzea.
- **Jarraipen eta kontrol egokia** izango da azken kalitate irizpidea. Proiektuaren prozesu osoan zehar eramango da egindako aurrerapenen kontrola eta ataza bakoitzean dedikatu diren ordu kopurua, hasierako plangintzarekin bat egin dezan ahal den heinean behintzat. Kontrol hau kalkulu taula baten bitartez eramango du aurrera egileak, proiektua lantzeko erabiltzen den egun bakoitzeko, ataza bakoitzean erabiltzen diren ordu kopuruak erregistratuz.

2.7 Komunikazio eta informazio sistemak

Proiektua garatzeko *Binovo* enpresak ezarritako konputagailua erabiliko da, pandemia egoera dela eta, proiektua telelanez eramango baita aurrera. Dokumentazioari dagokionez ordea, ordenadore pertsonalaz gain, *Google Drive* ere erabiliko da. Alde batetik, informazio guztia leku bat baino gehiagotan gorde eta edozein galera ekiditeko; eta beste aldetik, proiektuko zuzendariarekin materiala modu eroso eta seguru batean elkarbanatu eta egindako lanaren jarraipena egiteko. Gainera, *Overleaf* editoreaz baliatuko gara dokumentazioaren osaketarako. Online edizioa eskaintzen duelako eta *LaTeX* testu dokumentuak sortzeko sistema erosoa delako.

Komunikazioari dagokionez, pandemia egoera dela eta ezinbestekoa izango da internet bidezko komunikazio sistema konfiantzazko bat izatea. Gradu Amaierako Lan honen zuzendariarekin eposta baliatuko da komunikazio laburretarako eta jarraipena egiteko, baita *BBC* edo *Google Meet* telebileretarako. Lanketa sakonago baten beharra dagoenean, aurrez aurreko bilerak egingo dira Informatikako Fakultatean. Enpresako tutorearen eta kideekin berriz, komunikazio guztiak online izango dira, eposta eta *Google Hangouts* bitartez zalantza eta mezu laburretarako eta *BigBlueButton* bidez bilera edo azalpen luzeagoetarako bideodei bitartez.

2.8 Interesatuak

Gradu Amaierako Lan honen interesatuak identifikatzerako momentuan, esan behar da lehen interesantua proiektuaren egilea bera dela. Egilearen ardura izango da proiektua arrakastaz burutzea ahalik eta modu bikainenean; eta gainera, aukera ezin hobea izango da alde batetik akademikoki garatzeko, baina baita lan-mundura gerturatzeko ere.

Interesatuen artean ere *Binovo* enpresa ere aipatu behar da, beraiek proposatutako proiektua baita, eta etorkizunean proiektu honetatik aterako den produktua enpresan bertan erabiltzeko interesa baitute. Proiektuaren zuzendaria, kasu honetan Rosa Arruabarrena, eta defentsako epaimahaia ere interesatuak izango dira. Zuzendariak proiektuaren garapenean gidatu eta zuzenketak egin beharko dituelako Gradu Amaierako Lan zuzen bat izan dadin, eta epaimahaiaren ardura izango delako egindako lana ebaluatzea.

Azkenik, dagoeneko *Elkarbackup* programaren erabiltzaileak ere interesatuak direla esan daiteke. Azken ginean, software libreko programa bat izanik, behin emaitza dagoenean erabilgarri izan daiteke hauetako askorentzat, eta nahi bezala erabili dezakete.

3

Teknologiak

Atal honetan proiektua aurrera eramateko erabili diren teknologiak aurkeztuko dira. Teknologia bakoitza zertan datzan azaltzeaz gain, hau aukeratzeko arrazoia azalduko da (egon bada) eta zer funtzio betetzen duen proiektu honen baitan.

PHP

PHP [20] (PHP: Hypertext Preprocessor) kode irekiko programazio-lengoaia interpretatua da. Honen erabilera oso zabaldua dago eta bereziki egokia da web garapenerako. Gainera, HTML markaketa lengoaian txerta daiteke, izan ere, hasiera batean web-orrien dinamismoa handitzeko diseinatua izan zen.

PHP kodea zerbitzarian exekutatzen den script lengoaia bat da, bezeroari HTML lengoaiako emaitza bidaliko diona, jatorrizko kodea agerian utzi gabe.

Symfony

Symfony [21] PHP lan-ingurune bat da. Lan-ingurune bat dela esatean bi osagai nagusi hartu behar dira kontuan. Batetik, eskaintzen dituen aurrez eraikitako software osagaiak, norberak eraikitako kode kantitatea murriztuko duena, baita akatsak ekidin ere. Eta bestetik, metodologia; aplikazioen egitura zehazten du. Honek hasiera batean aukerak mugatzen dituela irudi dezake, baina proiektuak modu eraginkor eta efektiboagoan garatzean laguntzen du.

Elkarbackup Symfony lan-ingurunean dago eraikia hasieratik. *Elkarbackup* aplikazioren azken bertsioa Symfony 4.4 bertsioa dago garatua, eta honen gainean inplementatu beharko da APIa.

API Platform

Beste aukerak

Elkarbackup aplikaziorako API-a eraikitzeko lehen pausoa hau egiteko erabiliko den tresna aukeratzea izan da. Hiru aukera posible planteatu dira:

Symfony Jadanik Symfony lan-inguruneak eskeintzen dituen ruta eta kontroladore sistemekin, berez ez litzateke eragozpenik egongo API-aren sarrera puntu bakoitzeko kontroladore espezifikoak inplementatu eta hauek erabiltzeko.

FOS Rest Bundle Symfony-ren *bundle* edo pakete bat da[1], duela urte batzuk arte hau zen APIak sortzeko aukera ohikoena, eta gaur egun ere erabiltzeko aukera egongo litzateke. Symfony bakarrik erabiltzearekin alderatuta *bundle* honek hainbat atal erraztuko lituzke, baina antolaketa mailan oso antzekoa da. Aukera honekin dokumentazioa sortzeko beste *bundle* batekin konbinatu beharko litzateke, *Nelmio Api Doc Bundle*[2] adibidez.

API Plarform

API-a zerotik sortzeko kontroladore edo lan-ingurune bat erabiltzea komeni da. Horregatik, eta beste aukerekin alderatu ondoren, proiektu honetarako API Platform erabiltzea erabaki da.

Izenetik ondorioztatu daitekeen bezala, APIak garatzeko "plataforma"edo lan-ingurune bat da hau. Symfonyren osagaien gainean APIak eraikitzeko pentsatuta dago, baina ez da soilik Symfony-ren kontroladoreen gainean konfiguratzen edo anotatzen, baizik eta APIarentzako arkitektura propioa zehazten du, *Jamstack* arkitektura hain zuzen.

Hau horrela, APIetan egiten diren *boilerplate*[3] ataza gehienak kontuan hartzen ditu API Platform-ek, dokumentazio sorrera barne. Gainera, dokumentazioa APIa deskribatzeko sortutako POPOetatik (Plain Old PHP Object) eraikiko da automatikoki; horretarako OpenAPI erabiliz[4]. Beraz, dokumentazioa eskuz edo beste bundle batek egitea baino fidagarri eta erosoagoa izango da.

Azkenik, aipatu API Plarformek *GraphQL* eta *Next.js*(kodearen sorrera automatizatzeko aukera ematen du) integratuak dituela. Hala ere, printzipioz ez dira tresna hauek erabiliko.

JSON

JSON[6], edo *JavaScript Object Notation*, datuak elkarbanatzeko formatu arin bat da. Gizakiarentzako erraza da bai idaztea bai irakurtzea, eta makinarentzat interpretatu eta sortzeak ez du zailtasunik suposatzen.

JSON bi estrukturek osatzen dute. Batetik, izena/balio bikote multzo batek; bestetik, balioen lista ordenatu batek. Bi estruktura hauek unibertsalak dira eta programazio lengoaia guztiak dira hauek tratatzeko gai; beraz, zentzua du lengoaiarekiko independentea den formatu batek bi egitura hauek izatea oinarrian.

Proiektu honetan garatuko den API-ak HTTP operazioak izango dituenez oinarri, datuen zati handi bat JSON formatuan kudeatu eta elkar-banatuko da: bai informazioa bidaltzeko baita informazioa jasotzeko ere.

[1] FOS Rest Bundle, https://symfony.com/doc/current/bundles/FOSRestBundle/index.html
[2] Nelmio Api Doc Bundle, https://github.com/nelmio/NelmioApiDocBundle
[3] ***Boilerplate***[2] hainbat lekutan errepikatzen den kode-zatiei deritzo, elkarren arteko ia ezberdintasunik ez dutenak.
[4] OpenAPI, https://www.openapis.org/

Box Project

Box Project[3] *PHAR* fitxategiak eraikitzeko aplikazioa da. Hainbat aukera eskeintzen ditu: aplikazioen trinkotze azkarra, PHAR fitxategien inguruko informazioa eman, betekizunen kontrola eta beste hainbat funtzionalitate. Gradu Amaierako Lan honetan, garatuko den komando-lerroko bezeroa fitxategi exekutagarri bakarrean biltzeko erabiliko da.

Git

Git[5] bertsio-kontrolerako doaneko eta kode irekiko softwarea da. Garapen prozesuan zehar egindako aldaketen aztarnen jarraipena egitea ahalbidetzen die garatzaileei, bai proiektu txikietan, baita handietan ere.

Bertsio-kontrolari dagokionez, badaude hainbat alternatiba, baina Git da *Binovo* enpresak erabiltzen duena, eta beraz, aurrez ezarritako teknologia bat izan da proiektu honetarako. Gainera, egileak aurrez erabili izan du software hau bertsio-kontrolerako eta ez du eragozpenik suposatu.

Eclipse

Eclipse[15] kode irekiko software plataforma bat da, garapenerako ingurune integratu bat. Batez ere *Java* aplikazioak garatzeko erabiltzen da, baina beste programazio-lengoaia askotan garatzeko ere balio du, kasu honetan PHP lengoaia erabiliko den bezala.

Ingurune hau erabiltzea erabaki da batik bat enpresako garatzaile gehienek hau erabiltzen dutelako eta gomendatua izan delako; eta gainera, proiektu honen egilea aurretik plataforma honekin aritua denez, ezaguna eta erosoa duelako.

EGit

Eclipserako Git integraziorako plugina da **EGit**[16]. Plugin honen bidez Git biltegiak irakurri, idatzi eta kudeatzeko aukera dago Eclipse ingurune bertatik. Honek garapena asko erraztu eta erosoagoa izango da, esan bezala, bertsio-kontrola Git bidez egitea erabaki delako eta proiektua Eclipsen garatuko delako.

4

Aurrekariak

Proiektuaren helburua ez denez aplikazio bat zerotik eraikitzea, baizik eta *Elkarbackup* softwarea zabaltzea aurretik zegoenari eraldaketak eginez, honen garapenean murgildu aurretik ezinbestekoa izango da jada daukaguna ondo aztertu eta bere funtzionamendua ulertzea; horrela, beharrezko atalak identifikatu eta hauek berrerabiliz RESTful APIa garatzeko. Beraz, atal honetan *Elkarbackup* aplikazioaren xehetasunak azalduko dira. Gainera, garrantzitsua da API-ak izan beharko dituen ezaugarriak argi izatea; esan bezala, APIak REST arkitektura izango du eta horretarako atal honen bigarren zatian aurkeztuko dira *RESTful API* baten nondik norakoak ere.

4.1 *Elkarbackup*

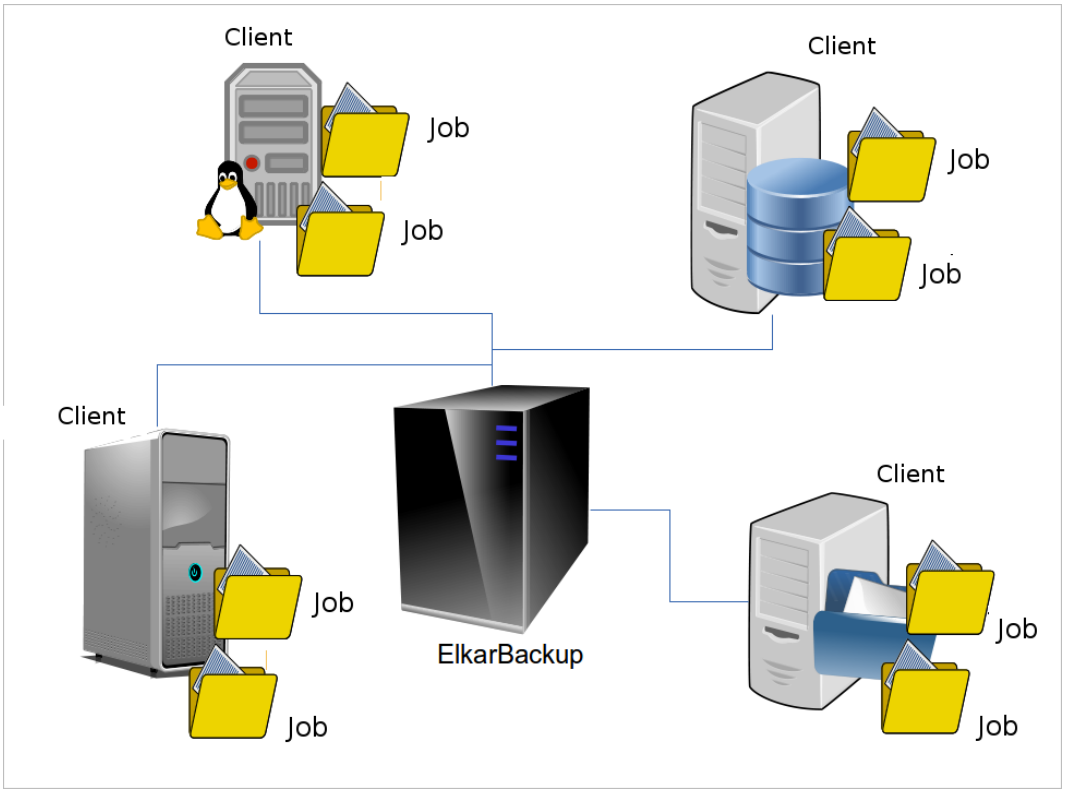
Elkarbackup[13] segurtasun-kopiak kudeatzeko kode-irekiko sistema erabilerraz eta indartsua da. Bezero-zerbitzari arkitektura erabiltzen du eta edozein sistema eragile duten makinen kopiak egin ditzake, bai sare-lokalean atzigarri daudenak, bai beste sareetakoak ere; azken kasu honetan, VPN edo ssh bezalako teknologiak erabili beharko litzateke makinarekin konexioa ezartzeko. Dena den, segurtasun-kopiak kudeatuko dituen zerbitzariak *Linux* sistema eragilea izan beharko du. *Linux* makinan instalatutako zerbitzaritik segurtasun-kopiak egiteko, *RSnapshot* eta *RSync* kode-irekiko teknologietaz baliatzen da.

Teknologiak

RSnapshot[9] fitxategi-sistemen irudiak sortzeko tresna bat da, bai makina lokaletan, baita urrutiko makinetan ere, *ssh* teknologiaren bitartez. Tresna honek esteka gogorrak erabiltzen ditu aukera dagoen bakoitzean, horrela diskoaren edukieraren erabilera asko txikituz. Tresna hau bere horretan erabiltzean, komando-lerrotik kontrolatzen da, baina *Elkarbackup*-ek abstrakzio maila bat eskaintzen du, hau web-interfazetik kudeatu ahal izateko.

RSnapshot tresna ***RSync***[10] sisteman oinarritzen da. Sistema honek fitxategiak sare bateko bi makinen artean edo makina bakarreko bi kokapenen artean bidali eta sinkronizatzeko balio du. Normalean, fitxategiak saretik trukatzeko *ssh* protokoloa[1] erabili ohi da.

[1] SSH protokoloa, https://eu.wikipedia.org/wiki/Secure_Shell



4.1 Irudia: *Elkarbackup*-en egitura

Egitura

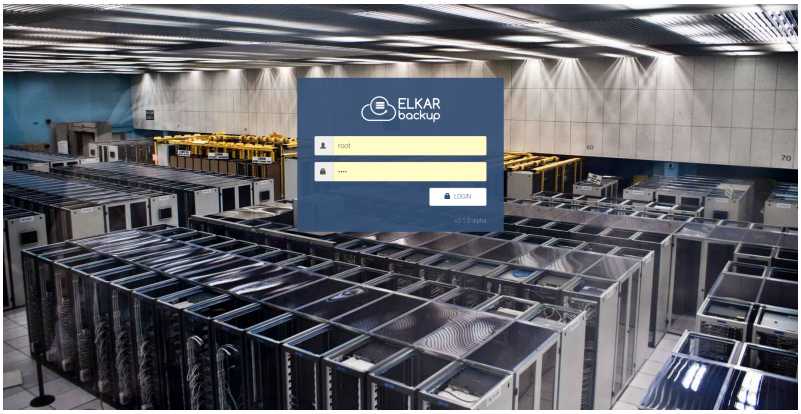
Zerbitzarian segurtasun-kopiak konfiguratu eta kudeatzeko, **bezero** (*client*) eta **lanak**(*job*) definitzen dira, 4.1 irudian ikus daitekeen bezala. **Bezero** bat sortzean, zerbitzariaren sare-lokaleko makina bati egiten dio erreferentzia. *Ssh* edo *rsync* bidez konektatzeko aukera ematen duen edozein makina izan daiteke.

Jarraitu aurretik, terminologia aldetik bezero eta web-bezero kontzeptuak bereiztea komeni da. Web-bezero deituko zaio *Elkarbackup* aplikazioa instalatu eta kontsumituko duen makinari, eta bezero izango dira aurretik aipatu bezala segurtasun-kopia egin nahi zaien makina oro.

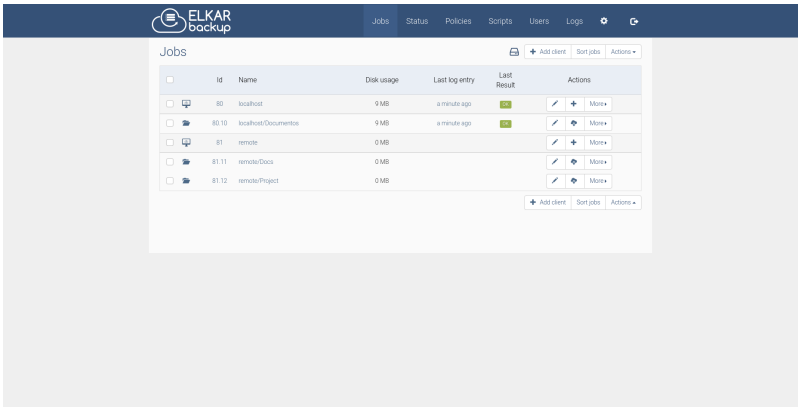
Bezero bakoitzean hainbat lan defini daitezke. **Lan** bakoitza bezerotik gorde nahi den direktorio bat izango da, eta konfigurazio propioa izango du, bere maiztasun, politika, eta segurtasun-kopiaren kokapenarekin.

Aipatu bezala, lan bakoitzak bere *politika* izango du. Politika horretan zehaztuko da segurtasun-kopia egikaritzeko maiztasuna, atxikitze-politika (*retention policy*) eta fitxategi eta karpetak baztertu edo sartzeko aukera. Politika hauek aurrez defini daitezke, lan ezberdinetan berrerabiltzeko, eta bakoitza bere izena eta deskribapenarekin gordeko da.

Bestalde, *Elkarbackup*-ek *script*-ak exekutatzeko aukera ematen du segurtasun-kopiak egikaritzerako orduan. *Script* bakoitza bi ezaugarriren arabera sailka daiteke: ea bezero mailakoa edo lan mailakoa den; eta ea segurtasun-kopia egin aurretik edo ondoren exekuta daitekeen (*postScript* edo *preScript* deituko zaio). Posible da aukera guztietarako egokia



4.2 Irudia: Login interfazea



4.3 Irudia: Orri nagusia

izatea. *Script* hauek ere aurrez gordeko dira nahi denean erabili ahal izateko.

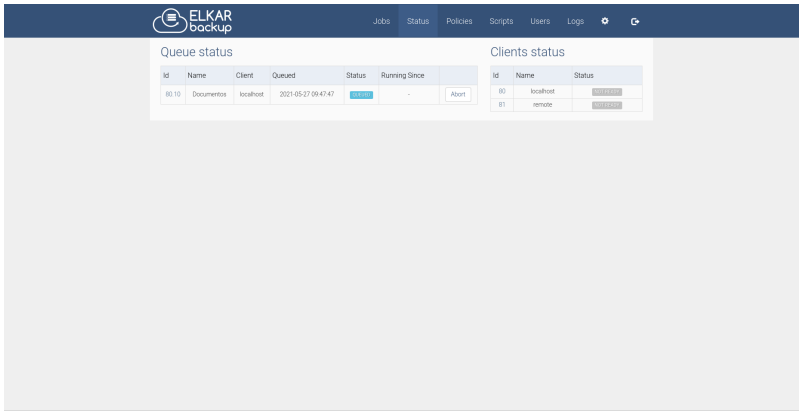
Aplikazio hau web-interfaze baten bitartez kudeatzen da. Interfaze hau hainbat hizkuntzetan eskaintzen da; euskaraz, gaztelaniaz, ingelesez eta alemanez, hain zuzen.

Jarraian ikusi daitezke kautotze-orria (4.2 irudia), non erabiltzaile eta pasahitza sartu behar den formulario soil batean; hasierako orri nagusia (4.3 irudia), bertan azaltzen da erregistratutako bezero eta lanen zerrenda, bakoitzaren egoera eta ekintzekin; ilara-orria (4.4 irudia), babes-kopiak egikaritzeko lanen ilara edo zerrenda, bakoitzaren egoera adierazita; azkenik, bezeroen xehetasunak gordetzeko formularioa (4.5 irudia), lanena ere antzeko formulario bat da, dagozkion atributuekin.

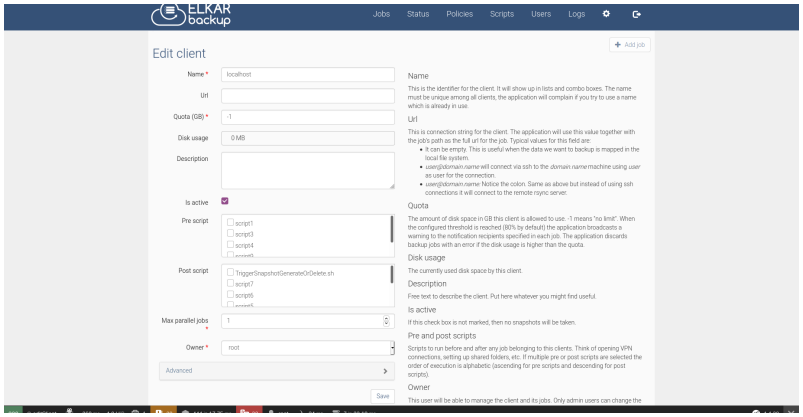
4.2 RESTful API

Hasieratik aipatu da Gradu Amaierako Lan honen helburu nagusietako bat RESTful API bat garatzea dela. Hori horrela, ezinbestekoa da argi izatea zer den REST arkitektura, zer ezaugarri dituen eta zein baldintza bete beharko dituen.

REST, edo *Representational State Transfer*[7][8] ingelesez, web-a bezalako hipermedia-sistema banatuetarako software-arkitektura estilo bat da. Kontzeptu hau 2000 urtean erabili zuen Roy Fielding informatiko estatubatuarrak, HTTP protokoloaren sortzaileetako batek,



4.4 Irudia: *Elkarbackup* ilara orria



4.5 Irudia: *Elkarbackup* bezeroaren galdetegia

lehen aldiz bere doktorego tesian. REST arkitekturak ezarritako baldintza eta printzipioak betetzen dituen aplikazioari, ***RESTful*** dela esango zaio.

Proiektu honetan garatuko den API-a RESTful izan dadin, honako printzipio hauek bete beharko ditu:

1. **Bezero-zerbitzari arkitektura.** Ardurak banatuz, eramangarritasuna ahalbidetzen du plataformen artean eta eskalagarritasuna hobetu zerbitzari aldeko atalak sinplifikatzerakoan.

2. ***Stateless* edo egoerarik gabea.** Bezerotik zerbitzarirako eskaera bakoitzak eskaera ulertzeko informazio guztia izan beharko du, ezingo da zerbitzarian gordetako testuinguruaren informaziorik erabili. Hau da, ezingo da saioen kontrolik egin.

3. ***Cachea* erabiltzeko aukera.** Eskaeren erantzunak *cachean* gordetzeko aukera dago. Aukera honek bezero-zerbitzari arteko elkarrekintzaren bat ekidin dezake, eskalagarritasuna eta errendimendua hobetuz.

4. **Interfaze uniformea.**

    - **Eskaerako baliabideen identifikazioa.** RESTful web-zerbitzuetan, esaterako, URI propio baten bidez identifikatuko litzateke baliabide bakoitza.

- **Baliabideak errepresentazioen bidez manipulatzea.** Bezero batek baliabide baten errepresentazioa badu, atxikitako metadatuak barne, nahiko informazio izango du baliabide hau eraldatu edo ezabatzeko. Hau da, bezeroa aurretik GET operazio baten bidez jasotako baliabide baten gainean egindako operazio baten erantzuna aurreikusteko gai izango da.
- **Mezu autodeskribatzaileak.** Mezu bakoitzak informazio nahikoa eskaini behar du bezeroak uler dezan. Ez luke informazio gehigarria behar dokumentazioa bereizi batean, adibidez.
- **Hipermedia aplikazioaren egoeraren motor gisa.** Behin aplikazioaren URI nagusira sartuta, REST bezero batek gai izan beharko luke zerbitzariak dinamikoki emandako esteken bidez nahi dituen baliabideak aurkitzeko.

5. **Sistema mailakatua.** Arkitektura modu hierarkikoan antolatzea ahalbidetzen du, mailen arteko komunikazioari segurtasuna eta eraginkortasuna gehituz.

6. ***Code on demand.*** Zerbitzariak bezero-aldeko funtzionalitate bat moldatu edo zabaltzeko gai izango da kode zatiak igorriz.

REST aplikazioen ezaugarri nagusiak azalduta, azpimarratu arkitektura diseinu honek zerbitzari eta bezeroen arteko informazioaren partekatze prozesua modu efikaz batean egitea ahalbidetzen duela, eta hau dela Binovok *Elkarbackup*-en API-arentzat estilo hau aukeratzeko arrazoi nagusia.

5

Soluzioaren diseinua

API Platform-ek garapenerako bi aukera ditu lan-ingurune bezala: *design-first* edo lehenengo diseinatu eta ondoren diseinatutako eraikitzea; edo *code-first*, hau da, kodea eraiki ahala forma ematen joan. Hala ere, lehenengo diseinua zehaztu eta ondoren kodetzea da gomendioa.

Atal honetan hiru bloke nagusi aurkeztuko dira. Batetik, API-arentzat zehaztu den diseinua, horretarako API Platformek zehazten duen arkitektura azalduz eta inplementatuko diren erabilpen-kasuen xehetasunak aurkeztuz. Bestetik, API-a frogatzeko aukeratu diren test motaren azalpena eta hauen diseinua. Eta azkenik, API-a kontrolatzeko garatuko den komando-lerroko bezeroaren inguruko xehetasunak.

5.1 *Elkarbackup* RESTful API

Esan bezala, proiektu honen lehen atal nagusia *Elkarbackup* aplikazioaren RESTful API-a garatzea da API Platform lan-ingurunean. Lan-ingurune honek, API-ak izango duen arkitektura finkatzen du, jarraian azalduko den moduan, eta behin hau zehaztuta, API-ak izango dituen erabilpen-kasuak eta kautotze metodoa diseinatuko dira.

5.1.1 Arkitektura

API-a diseinatzerakoan lehen pausoa da API-aren muturretako egitura publikoa definitzea. Horretarako sarrera eta irteerako *DTO*-ak definituko dira.

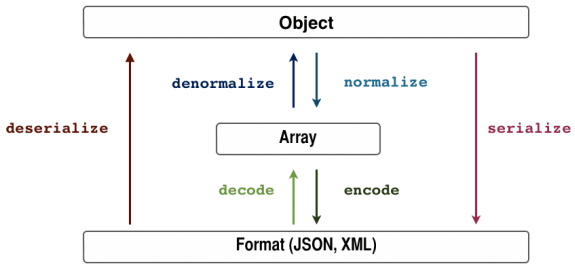
DTO (*Data Transfer Object*)

Prozesuen artean informazioa garraiatzeko balio duten *POPO*-ak dira, hau da, *plain old PHP object* edo PHP lengoaiako objektu sinpleak, hau da, inongo klase hedapenik behar ez dutenak. Entitateak berak funtziona dezakete DTO bezala, baina sarrera eta irteerako informazioa egokitu nahi bada, DTO ezberdinak definitzea komeniko da.

Data Transformer

Sarrera eta irteerako DTO horiek entitatearen ikuspegi zehatz bat aurkeztuko dutenez, Data Transformerrak DTO hauek entitate bihurtzeko eta alderantzizko prozesua ahalbidetuko duten klaseak dira. Horrela, sarrerako DTO-tik entitatea sortu edo lortzean ***deserializazio***

prozesua emango da, eta entitatetik irteerako DTO-a sortu eta erabiltzaileari *JSON* fitxategia itzultzean ***serializazio*** prozesua emango da (ikus 5.1 irudia). Behin sarrerako JSON



5.1 Irudia: Serializazio eta deserializazio prozesua

fitxategitik *client* edo bezero entitate bat lor daitekeenean eta alderantziz, entitate batetik irteerako JSON informazioa itzul daitekeenean, informazio hau lortu eta gordetzeaz arduratuko diren klase edo objektuak beharko dira. Eginkizun hori izango dute *data provider* eta *data persister*-ak.

Data Provider

API-ak azaleratuko duen informazioa lortzeko klaseak dira *data provider*-ak. Informazioaren jatorriaren arabera, egitura eta osagai bat edo beste izango dituzte klase hauek datuak eskuratzeko, esaterako, *Doctrine ORM*, *Doctrine MongoDB ODM* edo *Elasticsearch-PHP*. Gradu Amaierako Lan honen kasuan datuan datu-base batetik atzitzen dira eta *Doctrine ORM* erabiliko da horretarako.

API Platform-ek Doctrine ORM-ren defektuzko *data provider*-a erabiltzeko aukera ematen du, baina interesgarriagoa da eta malgutasun handiagoa dago entitate bakoitza lortzeko klaseak sortzen badira; bat entitateen kolekzioa jasotzeko, eta bestea entitate zehatz bat lortzeko.

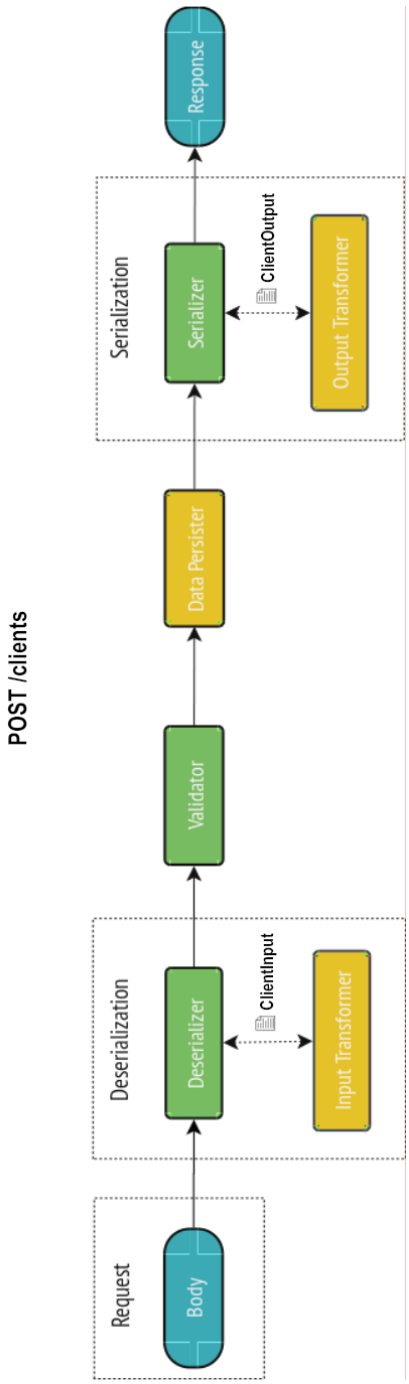
Data Persister

Datu-basean informazio berria gorde edo dagoeneko existitzen dena aldatzeko, *data persister* klaseak erabiliko dira; hau da, API-aren POST (5.2 irudia), PUT (5.3 irudia) eta DELETE eragiketetarako. *Data provider*-en kasuan bezala, *Doctrine ORM* erabiliko da eta mota honetako bi definituko dira: bat bezeroentzat eta bestea lanentzat.

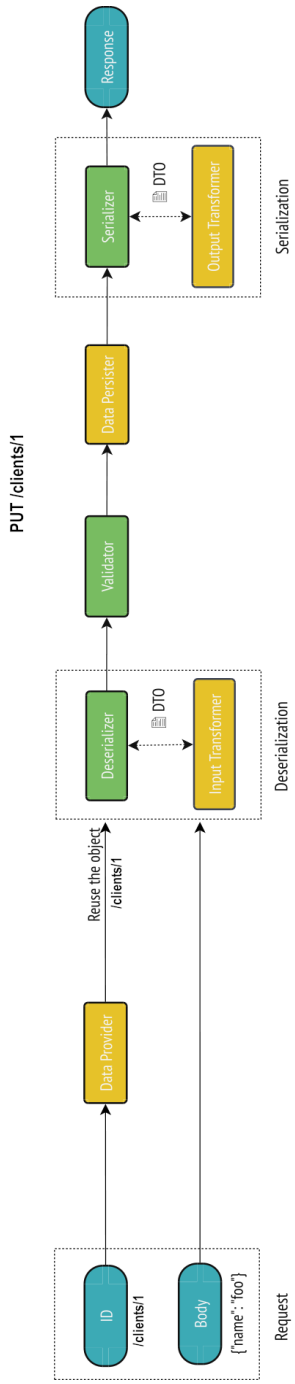
5.1.2 Egitura

Arkitektura ezagututa, eta bezero eta lanak hartuta entitate nagusi bezala, pixkanaka API-ak izango duen egiturari forma emateko momentua da. Honela, lau DTO izango dira, entitate bakoitzeko sarrerako bat eta irteerako bat; hau da, *clientInput* eta *clientOutput* bezeroen sarrera eta irteerarako, eta *jobInput* eta *jobOuput* lanen sarrera eta irteerarako. Hau horrela, DTO bakoitzarentzat *data transformer* bat inplementatuko da, DTO eta entitatearen arteko bihurketa egiteko.

Horrez gain, entitate bakoitzarentzat *data persister* eta bi *data provider* garatuko dira: elementu bakarra lortuko duena eta entitate multzoa (*collection*) itzuliko duena.



5.2 Irudia: POST operaziorako API Platform arkitektura



5.3 Irudia: PUT operaziorako API Platform arkitektura

Horrez gain, proiektuaren garapen-denbora errealaren arabera, hainbat entitate gehiago gehitzea interesgarria izango litzateke, bezero eta lanetatik erreferentziatzen baitira, 5.4 irudian azaltzen den bezala. Bertan ikusi daitezke API-tik atzitu daitezkeen bi entitate nagusiak, bezero eta lanak, eta baliabide horietatik erreferentziatzen diren politika, script, erabiltzaile eta babes-kopien kokapenak. API-aren irismena zabaltzeko aukera balego, denboraren kudeaketaren arabera betiere, API-tik politika, script eta beste entitateak atzitzeko aukera egongo litzateke.

5.1.3 Erabilpen-kasuak

Behin API-ak izan beharko duen egitura ezagututa, garatu beharko diren funtzionalitate edo erabilpen-kasuak aurkeztuko dira. 5.1 taulan ikus daitekeenez, bi multzo nagusitan sailkatzen dira erabilpen kasuak, hain zuzen, bezeroei dagozkienak eta lanak kudeatzeko izango direnak. Taulan bertan azaltzen da bakoitzaren azalpena, baita HTTP aginduak izango duen egitura ere. Gainera, 5.5 irudian ere ikus daiteke erabilpen-kasuen diagrama, aktore bakoitzak izango dituen baimen edo eskumenak bereizita.

Eredu bezala, bezero baten xehetasunak lortzeko erabilpen-kasua azalduko da. Honetarako hiru oinarrizko datu beharko dira: erabiltzailea, pasahitza eta interesatzen den bezeroaren identifikazio zenbakia. Erabiltzailea kautotzerakoan egiaztatuko da ea emandako id hori bezeroren bati egokitzen zaion, eta ea erabiltzaile horrek ze baimen mota dituen. Erabiltzaileak administrari baimenak baditu, berdin du bezeroaren jabea den edo ez, bezeroaren xehetasunak itzuliko ditu JSON formatuan. Erabiltzaileak ez baditu administrari baimenak ordea, erabiltzaileak eskatu den bezeroaren jabea izan beharko du, bestela errore mezu bat itzuliko da.

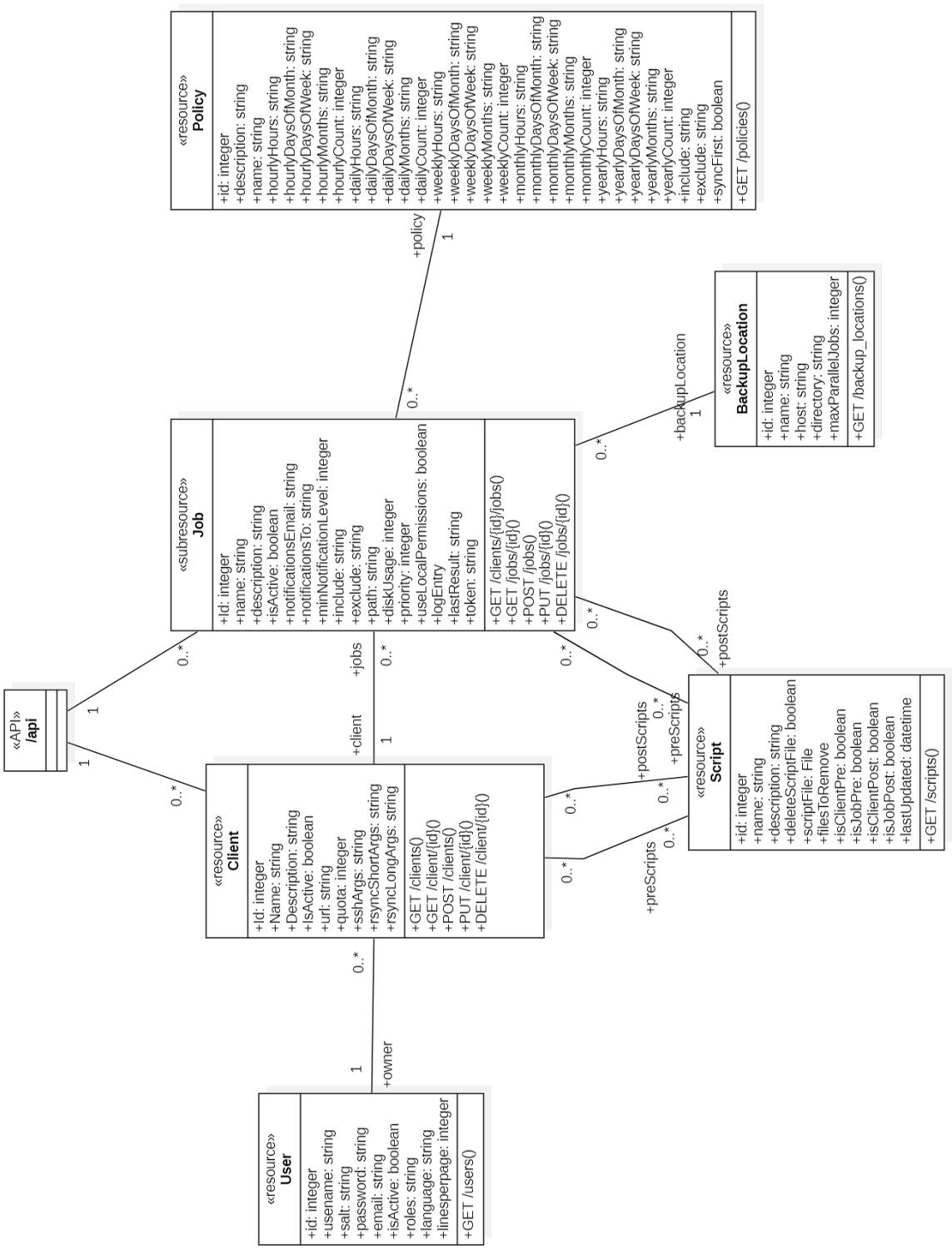
5.1.4 Autentifikazioa

Elkarbackup aplikazioaren kautotzea formulario bidez egin da orain arte. Metodo honekin, formularioa bidaltzean *cookie* bat gordetzen da, eta hau erabiltzen da erabiltzailea autentifikatzeko. Dena den, API batek *cookie*-ak erabiltzea ez denez oso zuzena, beste hiru aukera egokiago planteatu dira.

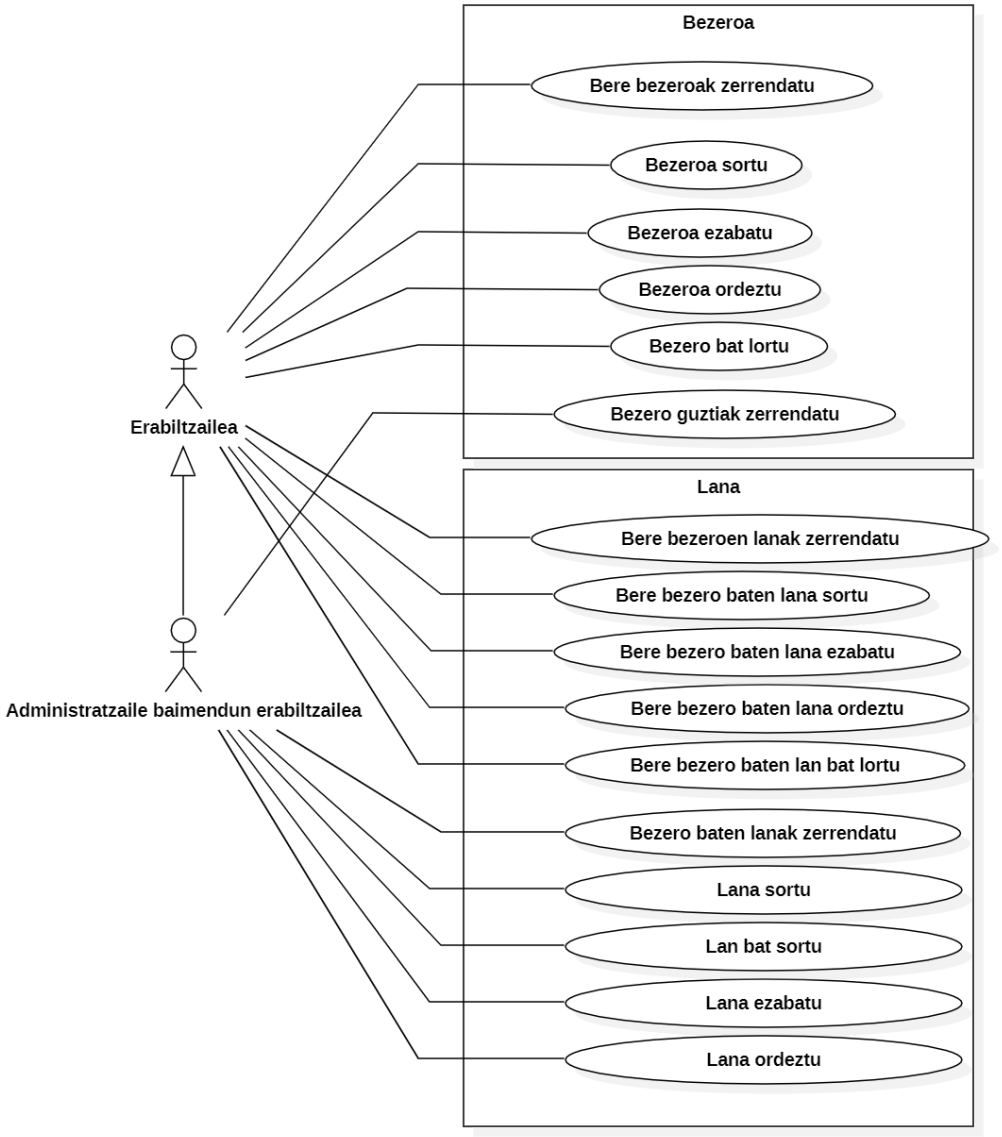
1. **JWT autentifikazioa**

   JWT, edo *JSON Web Token*, JSON-en oinarritutako estandar ireki bat da, informazioa alde batetik bestera modu seguru batean transmititzeko aukera ematen duena. Gehien bat autentifikazio eta pribilegio (edo *claims* ingelesez) datuak bidaltzeko erabiltzen da, azken finean, gako pribatu batez sinatutako JSON objektu bat besterik ez da izango. Estandar honekin kautotzeko sarrera puntu bat definituko litzateke, non erabiltzaile eta pasahitza bidaliko zen JSON fitxategi batean; APIak kredentzialak balidatuko lituzke, eta datuak zuzenak badira, JWT token bat sortuko luke, bere gako pribatuarekin sinatuta. Token horretan informazio asko bidali daiteke, baina kasu honetan nahikoa izango litzateke erabiltzailearen IDa eta tokenaren iraungitze data izatea. Modu honetan, API-aren hurrengo eskarietan *Authorization* HTTP burukoan tokena bidaliko luke. Aukera hau dagoeneko integratuta du API Platform inguruneak[1], beraz ez luke zailtasun asko suposatuko JWT autentifikazioa garatzeak API honetarako.

[1] API Platform JWT authentication, https://api-platform.com/docs/core/jwt/



5.4 Irudia: *Elkarbackup* API-aren klase-diagrama



5.5 Irudia: *Elkarbackup* API-rako erabilpen-kasuen diagrama

	HTTP metodoa	CRUD funtzioa	URI identifikatzailea	Azalpena
Bezero	GET	Read	/api/clients	Bezeroak zerrendatu
	GET	Read	/api/clients/{id}	Zehaztutako Id-a duen bezeroa eskuratu
	POST	Write	/api /clients	Bezero berria sortu
	PUT	Replace	/api/clients/{id}	Zehaztutako Id-a duen bezeroa, emandako datu berriekin ordeztu
	DELETE	Delete	/api/clients/{id}	Zehaztutako bezeroa ezabatu
Lan	GET	Read	/api/clients/{clientId}/jobs	*ClientId* bezeroaren lanak zerrendatu
	GET	Read	/api/clients/{clientId}/jobs/{jobId}	*ClientId* bezeroaren *jobId* lana eskuratu
	POST	Write	/api/clients/{clientId}/jobs	*ClientId* bezeroaren lan berria sortu
	PUT	Replace	/api/clients/{clientId}/jobs/{jobId}	Zehaztutako Id-a duen bezeroaren *jobId* lana, emandako datu berriekin ordeztu
	DELETE	Delete	/api/clients/{clientId}/jobs/{jobId}	*ClientId* bezeroaren *jobId* lana ezabatu

5.1 Taula: *Elkarbackup* API-rako operazioen notazio eta definizioa

2. **Login puntua**

   REST printzioetan azaldu bezala (4.2), RESTful API batek egoerarik gabekoa izan behar du, hau da, ezingo dira *session*-ak erabili erabiltzaileen egoerari buruzko informazioa jasotzeko. Horregatik, *cookie* bidezko autentifikazioa ez litzateke egokia izango API baterako. Hala ere, aukera moduan API token-ak erabiltzeko aukera egongo litzateke, horrela REST izaera mantenduko luke eta.

   Modu sinple batean azalduta, kautotzerako sarrera puntu bat definituko litzateke, eta erabiltzaileak emandako datuekin *HTTPOnly cookie* bat itzuliko litzateke, ondoren eskaera bakoitzean bidaliko litzatekeena bezeroa kautotzeko. *Cookie* hau ez da *JavaScript*-ez irakurgarria izango, beraz beste JavaScript scriptetatik babestua egongo da eta seguruagoa izango da.

   Dena den, esan daiteke hau ez dela oso aukera egokia, izan ere, WEB-erako pentsatutako metodo bat da, ez API baterako, eta mugak izan ditzake kontsumitzailearentzat, erabilitako mekanismoak *cookie*-ak kudeatzeko ahalmena izan behar baitu.

3. **HTTP basic**

   Planteatutako azken aukera, eta garatzea erabaki dena, ***HTTP basic*** kautotzea da. *HTTP basic* metodoak erabiltzaile-agente (*user agent*) batek, nabigatzaile batek adibidez, eskaera baterako erabiltzaile eta pasahitza eskatzea ahalbidetzen duen teknologia da. Oso metodo sinplea da, baina ez da oso segurua, eskaera bakoitzaren burukoetan

bidaltzen baita kautotze informazioa. Hori dela eta, *https* protokoloa erabiltzea gomendatzen da, paketeak enkriptatuta bidali eta burukoak irakurtezinak izan daitezen.

API Platformek ez du errezetarik aurkezten metodo honetarako; baina oinarrian Symfony-ren autentifikazio metodoak erabili besterik egiten ez duenez, ez da konplexua hau aurrera eramatea. Horretarako, aplikazioan API-arentzat *firewall* berri bat definituko da, hau *HTTP basic* autentifikazioa eskatuko duena. Autentifikazio parametroak eskaera bakoitzean bidali beharko litzateke, baina esan bezala, aukera sinplea da eta aldi berean API honek dituen beharretarako egokia eta nahikoa, ez da metodo konplexuago baten beharrik ikusi.

5.2 API-arentzat test automatikoak

API-aren funtzionamendua frogatzeko, testak diseinatzerako orduan, lehenik eta behin test mota bat finkatu, hau garatzeko azpiegitura edo tresnak aztertu eta, behin hori zehaztuta, testen xehetasunak definituko dira.

Esan bezala, lehen pausoa, test motak aztertu eta proiektu honetarako interesgarriena aukeratzea da. Software testak bi multzo nagusietan sailkatzen dira: funtzionalak eta ez-funtzionalak. **Proba ez-funtzionalak**, aplikazioaren funtzionamendua epaitzeko zehazten diren betekizunak egiaztatzean datza, hala nola, irisgarritasuna, mantengarritasuna, segurtasuna eta/edo errendimendua. Beste hitz batzuetan esanda, sistemak nola erantzuten duen egiaztatzen dute.

Test funtzionalek berriz, aplikazioak interesatuek zehaztutako jokabidea duela egiaztatzen dute, eta errorerik ez duela. Hauen barruan hainbat mota daude[17], baina erabilienak dira unitarioak, integraziokoak eta onarpenekoak deiturikoak. Test funtzionalak proiektu honetarako interesgarrienak izanik, jarraian azalduko dira xehetasun gehiagorekin.

5.2.1 Test motak

Unitarioak

Kodearen atal edo funtzionalitate txiki bat probatzeaz arduratzen dira. Honen helburua da kode-unitate zehatz horrek behar bezalako funtzionamendua duela egiaztatzea da. Kode-unitate hau orokorrean klase edo funtzio konkretu bat izango da. Funtzionamenduaz gain, kode-unitate horren izena, parametroen izen eta mota eta itzulerako balio eta mota ere egiaztatzen dira. Proba mota hauek oso azkarrak dira baina aplikazioaren oso atal txikia aztertzen dute, eta beraz ez dira nahikoak aplikazioaren funtzionamendua orokorrean zuzena dela egiaztatzeko.

Integraziokoak

Integrazio testek kode-unitate bat baino gehiagok elkarren artean funtzionamendu egokia dutela egiaztatzen da. Horretarako, oso ohikoa da test hauetan datu-basera dei bat edo HTTP eskaera bat burutzea. Proba hauek unitate, edo elementu ezberdinen arteko komunikazioa frogatzean oinarritzen da.

Onarpen-testak

Onarpen-testak, muturretik-muturrerakoak (*end-to-end*) edo aplikazio-testak ere deiturikoak, softwarea bere osotasunean probatzen dute, kanpoko interfazeekin integratzearekin batera. Proba hauekin menpekotasuna, integrazioa eta komunikazioa ere egiaztatzen da, funtzionalitateak modu orokor batean frogatuz.

Suposa daitekeenez, test mantsoenak dira. Baina aldi berean Gradu Amaierako Lan honi balio handiena eransten dioten probak dira, azken finean API-a modu integral batean probatzen duten test mota dira. Gainera, beste proba motaren bat garatzea erabakiko balitz, *Elkarbackup* aplikazioaren *legacy* kodearen probak ere garatzeko beharra egongo litzateke, neurri batean API-a honen gaineran eraikita baitago, eta horrek beste proiektu baterako lan-karga suposatuko luke. Labur esanda, mota honetako testak garatuta API-aren funtzionamendu zuzena berma daiteke, hauen inplementazio-denbora jasangarri batekin.

5.2.2 Praktika egokiak

Test egoki eta eraginkorretarako kontuan hartuko diren irizpideak:

- **Errepikagarriak** Proba bat behin jaurtitzean ez luke eragotzi beharko test hori bigarren aldiz jaurtitzea. Beste hitz batzuetan, elementu bat zuzen ezabatzen dela egiaztatzen duen testa bi aldiz exekutatzeko aukera egon behar du, ez du balio bigarrenean errore bat egotea dagoeneko elementua ezabatua izan delako lehen jaurtiketan.

- **Independenteak** Test baten emaitzak ezin du beste test baten exekuzioaren menpeko izan. Test guztiek nahi bezala funtzionatu behar dute, hau bakarrik exekutatuta, edo testak orden ezberdinetan exekutatuta, edo dena dela.

- **Esperotako emaitza** Funtzionalitatearen exekuziotik eskuratutako emaitza esperotakoa dela egiaztatu behar da, ez da nahikoa errorerik izan ez dela egiaztatzearekin.

5.2.3 Azpiegitura

Behin egingo diren test motak erabakita, hauen garapenerako erabiliko den azpiegitura edo tresna zehaztu beharko da. Kasu honetarako, garatutako API-a probatu nahi denez, eta hau *API Platform* lan-ingurunean eraiki denez, ingurune honek eskaintzen dituen *testing* tresnez baliatzea erabaki da, integritate eta dependentzi arazoak saihestera bidean.

API Platform-ek eskaintzen dituen testing funtzionalitateak *PHPUnit* lan-ingurunean oinarritzen dira. *API Platform*-ek *Symfony HttpClient*-en interfazeen berezko inplementazioa eskaintzen du, *PHPUnit* test klaseetan zuzenean erabiltzeko egokituta. Honek hobekuntza handia suposatzen du proben errendimenduan, benetako HTTP eskakizunekin alderatuta.

Probetarako datu-basea elikatu eta interesatzen den egoera bermatzeko, *fixture*ak [11] sortzeko aukera ere ematen du API Platform-ek. Fixtureak testentzat datuak modu erraz eta maiz ausaz sortzeko tresnak dira. Kasu honetan, *Alice*[1] eta bere *Symfony*-rako integrazioa (*AliceBundle*) erabiliko da datuen sorrerarako.

5.2.4 Diseinua

Onarpen-test bakoitza diseinatzeko *GivenWhenThen* [18] patroia jarraitu da. Hau da, lehenengo testa aurrera eramateko beharrezko prestakuntza pausoak eman dira(*given*); behin nahi diren baldintzak lortuta, frogatu nahi den ekintza burutzen da (*when*); eta azkenik, espero zen emaitza edo egoera eman dela egiaztatzen da (*then*).

Bezero eta lanen inguruko operazioen inguruko testen egitura nagusia 5.2 eta 5.3 tauletan zehaztu da; operazioa, frogatu nahi den egoera, espero den itzulerako kodea eta egiaztatu beharrekoa zehazki, eta zutabeka jarrita.

5.3 *Elkarbackup* API CLI

Komando-lerroko bezeroa (*Command Line Interface*) diseinatzerako orduan oinarrizko betekizun bezala jarri da interfazearen garapenerako teknologia eta egitura *Elkarbackup API*-aren ahalik eta antzekoena izatea, mantenua ahalik eta errazena izateko eta etorkizunean eguneraketa edo hobekuntzarik izanez gero, komando-lerroko bezeroaren eta API-aren arteko integritate eta bateragarritasun arazoak saihesteko.

Atal honetan komando-lerroko interfazea garatzeko erabiliko den osagaia azaldu, izango duen egitura diseinatu eta azken produktu exekutagarria lortzeko emango den paketatzea azalduko dira.

5.3.1 Symfony console

Aipatu bezala API-aren azpiegitura antzekoena erabiltzea komeni da etorkizuneko bateragarritasun arazoak ekiditeko. Hori dela eta, hasiera batean *PHP* soilean garatzea planteatu zen, baina aukerak sakonago ikertu ondoren, *Symfony*-ren *console* osagaia erabiltzea interesgarria dela ikusi da. Izan ere, *PHP* soilean garatzeak baina hainbat erraztasun gehiago ematen ditu, eta *Symfony* beraren osagaia izanik, API-aren eguneraketekin batera joateko aukera ematen du.

5.3.2 Komando-lerroko komandoen egitura

Behin komando-lerroko bezeroa garatzeko azpiegitura zehaztuta, kontsolak eta komandoek izango duten egitura zehaztea izango da hurrengo pausoa. Aurrerago azalduko diren moduan, 14 komando izango dira denera, bezero eta lanak sortu eta eguneratzeko bi aukera egongo baitira: JSON fitxategi bat bidalita informazioarekin, edo datuak parametro moduan pasata.

Komando guztiek parametro komun batzuk izango dituzte, bakoitzaren funtzionalitateei dagozkienez gain, hauek izanikdira kautotzerako erabiltzailea eta pasahitza, eta *HTTP* eskaerarako url-a konfiguratzeko parametroa. Gainera, bezero eta lanen datuak itzultzen dituzten komandoek *output* parametro bat izango dute irteerako informazioa JSON fitxategi batean gordetzeko aukera izateko. Jarraian aurkeztuko dira komandoak banan-bana.

`client:create:file`

Bezeroa sortuko du JSON fitxategi batean bidalitako datuekin. Horretarako izango du *input* parametroa. Fitxategia existitzen ez bada edo ez baditu beharrezko baimenak errore-mezu

Operazioa	Kasua	Erantzuna	Kodea	Egiaztapena
Zerrendatu (GET /clients)	Arrakastaz zerrendatu		200	Zerrendan bezero ezagun bat dago
Bakarra lortu (GET /clients/{id})	Arrakastaz lortu		200	Ezaguna den bezeroaren balioak zuzen itzultzen ditu
	Bezeroa ez da existitzen	NotFoundException	404	Errore mezua: “The client %s does not exist.”
Gorde & editatu (POST/PUT)	Arrakastaz gorde		200	Itzultzen duen bezeroak datu zuzenak ditu
	Izena existitzen da	BadRequest	400	Errore mezua: “Client name %s already exists”
	maxParallelJobs < 1	Unprocessable Entity	422	Errore mezua: “maxParallelJobs: Max parallel jobs value must be a positive integer”
	PreScriptak ez dira existitzen	InvalidInput	400	Errore mezua: “Script %s does not exist”
	PostScriptak ez dira existitzen	InvalidInput	400	Errore mezua: “Script %s does not exist”
	Scripta ez da client preScript	InvalidInput	400	Errore mezua: “Script %s is no a client pre script”
	Scripta ez da client postScript	InvalidInput	400	Errore mezua: “Script %s is no a client post script”
	Ownerra ez da existitzen	InvalidInput	400	Errore mezua: “Incorrect owner id”
Ezabatu (DELETE)	Arrakastaz ezabatu		204	Ezabatutako bezeroa ez dago zerrendan.
	Client-a ez da existitzen	NotFoundException	404	Errore mezua: “The client %s does not exist.”
	user!=owner & user.role!=admin	PermissionExcepton	403	Errore mezua: “Unable ti delete client: Permission denied.”
	jobsEnqueued	BadRequest	400	Errore mezua: “Could not delete client, it has jobs enqueued”
Editatu (PUT)	Bezeroa ez da existitzen	NotFoundException	404	Errore mezua: “The client %s does not exist.”

5.2 Taula: *Elkarbackup* API-ko bezeroen operazioen testen diseinua

Operazioa	Kasua	Erantzuna	Kodea	Egiaztapena
Zerrendatu (GET /jobs)	Arrakastaz zerrendatu guztiak		200	Zerrendan lan ezagun bat dago
	Arrakastaz zerrendatu bezero batenak (filtroa)		200	Bezero horren lan ezagun bat itzultzen du, datu zuzenekin
	Existitzen ez den bezero batekin filtratu		200	Zerrenda hutsa itzultzen du
Bakarra lortu (GET /jobs/{id})	Arrakastaz lortu		200	Lan ezagunaren datuak zuzen itzultzen ditu
	Bezeroa ez da existitzen	NotFoundException	404	Errore mezua: "The job %s does not exist."
Gorde & editatu (POST/PUT)	Arrakastaz gorde		200	Bidalitako datuekin itzultzen du gordetako lana
	Client ez da existitzen	InvalidInput	400	Errore mezua: "Incorrect client id"
	NotificationLevel okerra (0, 200, 300, 400, 1000)	InvalidInput	400	Errore mezua: "Incorrect notification level (0, 200, 300, 400, 1000)"
	PreScriptak ez dira existitzen	InvalidInput	400	Errore mezua: "Script %s does not exist"
	PostScriptak ez dira existitzen	InvalidInput	400	Errore mezua: "Script %s does not exist"
	Scripta ez da job preScript	InvalidInput	400	Errore mezua: "Script %s is no a job pre script"
	Scripta ez da job postScript	InvalidInput	400	Errore mezua: "Script %s is no a job post script"
	BackupLocation ez da existitzen	InvalidInput	400	Errore mezua: "Incorrect backup location id"
	NotificationsTo okerra	InvalidInput	400	Errore mezua: "Incorrect notifications to argument(owner, admin, email)"
	NotificationsEmail formatu okerra	InvalidInput	400	Errore mezua: "Incorrect notification email address"
	Policy ez da existitzen	InvalidInput	400	Errore mezua: "Incorrect policy id"
Ezabatu (DELETE)	Arrakastaz ezabatu		204	Ezabatutako lana ez dago lanen zerrendan
	Job-a ez da existitzen	NotFoundException	404	Errore mezua: "The job %s does not exist."
	user!=owner & user.role!=admin	PermissionExcepton	403	Errore mezua: "Unable to delete job: Permission denied."
	jobsEnqueued	BadRequest	400	Errore mezua: "Could not delete job %s, it is enqueued"
Editatu (PUT)	Lana ez da existitzen	NotFoundException	404	Errore mezua: "The job %s doe not exist."

5.3 Taula: *Elkarbackup* API-ko bezeroen operazioen testen diseinua

bat itzuliko du.

`client:create:manual`

Bezeroa sortuko du parametroetan txertatutako datuekin. Bezeroaren izena eta jabearen id-a ematea derrigorrezkoa izango da, beste informazioa guztia lehenetsitako balioekin edo hutsik bidali daiteke.

`client:delete`

Emandako *id* zenbakia duen bezeroa ezabatuko du, bezero hori existitzen bada.

`client:details`

Zehaztutako *id*-a duen bezeroaren xehetasunak itzuliko dira JSON formatuan. Hautazko *output* parametroa izango du, informazioa fitxategi batean gordetzeko.

`client:list`

Bezeroen zerrenda itzultzen du. *output* parametroarekin zerrenda JSON fitxategi batean gordetzeko aukera dago.

`client:update:file`

Derrigorrezko parametro moduan bidalitako *id* zenbakia duen bezeroa ordeztuko da *input* fitxategian emandako datuekin. Fitxategia existitzen ez bada edo ez baditu beharrezko baimenak errore mezu bat itzuliko du.

`client:update:manual`

Emandako *id* zenbakidun bezeroa ordeztuko da parametroetan bidalitako datuekin. Kontuan izan beharko da bezeroaren propietate guztiak ordezten dituela, beraz, aurretik duen daturen bat mantendu nahi bada, berriro ere bidali beharko da. Hala ere, oinarrizko parametroak izena eta jabea izango dira.

`job:create:file`

Lana sortuko du JSON fitxategi batean bidalitako datuekin, horretarako *input* parametroa izanik. Fitxategia existitzen ez bada, edo ez baditu beharrezko baimenak, errore mezu bat itzuliko du.

`job:create:manual`

Lana sortuko du parametroetan txertatutako datuekin. Lanaren izena, lana dagokion bezeroaren identifikazio zenbakia eta lanaren helbidea (*path*) ematea derrigorrezkoa izango da, beste informazioa guztia lehenetsitako balioekin edo hutsik bidal daiteke.

`job:delete`

Identifikazio zenbaki hori duen lana ezabatuko du, existitzen bada.

`job:details`

Zehaztutako *id*-a duen lanaren xehetasunak itzuliko dira JSON formatuan. Hautazko *output* parametroa izango du, irteerako informazioa fitxategi batean gordetzeko.

`job:list`

Lanen zerrenda itzultzen du. *output* parametroarekin zerrenda JSON fitxategi batean gordetzeko aukera dago.

`job:update:file`

Derrigorrezko parametro den identifikazio zenbakia duen lana ordeztuko da *input* fitxategian emandako datuekin. Fitxategia existitzen ez bada edo ez baditu beharrezko baimenak errore mezu bat itzuliko du.

`job:update:manual`

Emandako *id* zenbakidun lana ordeztuko da parametroetan bidalitako datuekin. Kontuan izan beharko da bezeroaren propietate guztiak ordezten dituela, beraz, aurretik duen daturen bat mantendu nahi bada, berriro ere bidali beharko da. Hala ere, oinarrizko parametroak izena, bezeroaren identifikazio zenbakia eta lanaren helbidea (*path*) izango dira.

5.3.3 *Elkarbackup* CLI aplikazioaren paketatzea exekutagarri gisa

Azkenik, komando-lerroko bezero hau *phar* exekutagarri bat bezala erabili eta barreiatzeko planteatu da. *PHAR* (*PHP Archive*) fitxategiak aplikazio eta liburutegien banaketa ahalbidetzen duen pakete-formatu bat da. PHP kode-fitxategiak eta beste hainbat baliabide elkartzen ditu fitxategi bakar batean.

Modu honetara, fitxategi bakar batean egongo dira bilduta komando guztiak, eta exekutagarriaren parametro moduan pasako da exekutatu nahi den komandoaren izena eta behar diren atributuak. Gainera, komando bakoitzaren xehetasunak erakusteko aukera ere izango du *help* moduaren bitartez.

Proiektu honetako *Symfony*-ko komando-lerroko aplikazioa *phar* fitxategi bihurtzeko, *Box Project*[2] tresna baliatuko da. Horretarako, tresnaren konfigurazio egokia eta *script* bat garatuko da paketatzeko beharrezko komandoak exekutatzen dituena.

[2]Box Project, https://github.com/box-project/box

6

Soluzioaren garapena

Behin soluzioaren diseinua aztertu eta finkatuta, garapenaren nondik norakoak azalduko dira. Diseinuaren moduan, hiru atal nagusitan bereiziko da: API-aren inplementazioa, hau probatzeko testen garapena eta komando-lerroko bezeroaren garapena. Atal bakoitzean aurkeztuko dira izandako zailtasun eta arazoak, hartutako erabakiak, diseinuarekiko egindako moldaketak eta kode-zati esanguratsuenak.

Teknologien atalean (3) aipatu bezala, kodea *gitHub* bidez kudeatu da, beraz bertan aurki daiteke kode guztia:

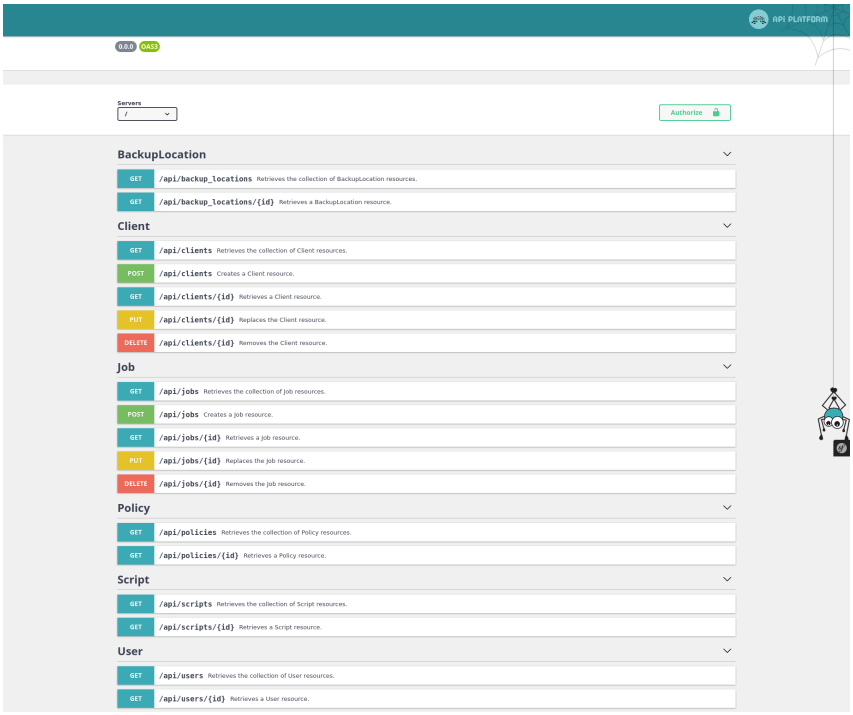
- *Elkarbackup* aplikazioa: `https://github.com/elkarbackup/elkarbackup`
- *Elkarbackup* API-a kontsumitzen duen komando-lerroko interfazea: `https://github.com/elkarbackup/elkarbackup-api-cli`

6.1 *Elkarbackup* API

API-a inplementatzerako orduan oso lagungarria izan da automatikoki sortutako dokumentazioa, bertan momentu oro API-a hartzen ari den egitura ikusiteko aukera baitago. DTO-ak, *data provider*-ak, *transformer*-ak, *data persister*-ak inplementatzen joan ahala API-a pixkanaka forma hartzen joan da. Gainera, kontuan hartu behar izan da kode-errepikapenak sahiestea, horretarako zerbitzuak garatuz. Atal honetan azalduko dira honen inguruko xehetasunak, baita izandako arazo eta moldaketak ere.

OpenAPI dokumentazioa

API-aren garapenean lehen pausoa entitate edo baliabideei forma ematea da, hau da, hasiera batean bezeroei eta lanei. Horretarako sarrera eta irteerako DTO-ak inplementatu dira, parametroak sartuz, motak zehaztuz, balio lehenetsiak emanez... eta hauek izango dituzten operazioak definituz. Garapen hau API-aren dokumentazioan ikus daiteke islatuta, 6.1 irudian azaltzen dira inplementatutako funtzionalitateak eta 6.2 irudian definitutako DTO-en eskemak, bai sarrera bai irteerakoak.



6.1 Irudia: *Elkarbackup* API-ren dokumentazioa

6.2 Irudia: *Elkarbackup* API-ren dokumentazioko DTO-en eskemak

DTO

Esan bezala, hau lortzeko, sarrera eta irteerako DTO-ak definitu behar izan dira. Esaterako, bezeroentzat *ClientInput* eta *ClientOutput POPO*-ak inplementatu dira. Script, erabiltzaile, babes-kopien lokalizazio edo politikentzat berriz, nahikoa izan da irteerako (*output*) klaseak inplementatzea, GET funtzioak bakarrik garatu baitira entitate hauentzat.

POPO klase hauek kasurako interesgarriak diren entitateen atributuak eta hauen *getter* eta *setter*-ak dituzten klase sinpleak izango dira. Adibidez, bezeroen informazioa eskatzean izena eta deskribapena jasotzea nahikoa balitz, orduan irteerako DTO-an bi atributu horiek bakarrik definituko lirateke. Jarraian politiken irteerako DTO-a adibide gisa.

```php
<?php
namespace App\Api\Dto;

class PolicyOutput
{
    private $description;
    private $id;
    private $name;

    /**
     * @return string
     */
    public function getDescription()
    {
        return $this->description;
    }

    /**
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }
```

Data Transformer

DTO-ak inplementatuta, hauen eta entitateen arteko bihurketak egiteko klaseak definitu behar dira, hau da, *dataTransformer*-ak. Hauek API Platform-en *DataTransformerInterface* interfazea inplementatuko dute. Jarraian txertatutako kodeetan ikus daiteke sarreratik entitatera egindako bihurketa:

```php
public function transform($data, string $to, array $context = [])
{
    if (isset($context[AbstractItemNormalizer::OBJECT_TO_POPULATE])) {
        $client = $context[AbstractItemNormalizer::OBJECT_TO_POPULATE];
    } else {
        $client = new Client();
    }
    $client->setName($data->getName());
    $client->setUrl($data->getUrl());
    $this->setQuota($client, $data->getQuota());
    $client->setDescription($data->getDescription());
    $client->setIsActive($data->getIsActive());
    $this->setPreScripts($client, $data->getPreScripts());
```

```
14      $this->setPostScripts($client, $data->getPostScripts());
15      $client->setMaxParallelJobs($data->getMaxParallelJobs());
16      $this->setOwner($client, $data->getOwner());
17      $client->setSshArgs($data->getSshArgs());
18      $client->setRsyncShortArgs($data->getRsyncShortArgs());
19      $client->setRsyncLongArgs($data->getRsyncLongArgs());
20      return $client;
21  }
```

Eta alderantziz, entitatetik, kasu honetan politika entitatetik, irteerako DTO-ra, *policyOutput* klasera bihurketa egiten duen klaseko funtzio nagusia.

```
1  public function transform($data, string $to, array $context = [])
2  {
3      $output = new PolicyOutput();
4      $output->setId($data->getId());
5      $output->setName($data->getName());
6      $output->setDescription($data->getDescription());
7      return $output;
8  }
```

Gainera hainbat froga egiten dira klase hauetan: mota egokia edo id zenbaki zuzena, esaterako. Adibidez, jarraian *JobInputDataTransformer*-eko *setClient* funtzioaren inplementazioa azaltzen da. Izan ere, lan baten datuak sartzean, honen bezeroa zehazteko identifikazio zenbakia ematen da eta lana gorde aurretik ziurtatu behar da bezero hura existitzen den edo ez.

```
1  private function setClient (Job $job, $clientId) {
2      $repository = $this->entityManager->getRepository('App:Client');
3      $query = $repository->createQueryBuilder('c');
4      $query->where($query->expr()->eq('c.id', $clientId));
5      if (null == $query->getQuery()->getOneOrNullResult()) {
6          throw new InvalidArgumentException ("Incorrect client id");
7      } else {
8          if (!$this->authChecker->isGranted('ROLE_ADMIN')) {
9              $query->andWhere($query->expr()->eq('c.owner', $this->
       security->getToken()->getUser()->getId()));
10             if (null == $query->getQuery()->getOneOrNullResult()) {
11                 throw new PermissionException(sprintf("Permission denied
       to create job of client %s", $clientId));
12             }
13         }
14         $job->setClient($query->getQuery()->getOneOrNullResult());
15     }
16 }
```

Data Provider

Datu-basetik informazioa lortzeaz arduratuko diren klase hauek bi multzotan bereiz daitezke: baliabide bakarra lortuko dutenak, hauek *ItemDataProviderInterface* interfazea inplementatuko dute, eta bilduma osoa itzuliko dutenak, *ContextAwareCollectionDataProviderInterface* inplementatuz. Dena den, *data provider* guztiek *RestrictedDataProviderInterface* interfazea ere inplementatuko dute, bakoitza zein entitatez arduratuko den zehazteko. Ondorioz hauek dira inplementatutako klaseak:

- BackupLocationCollectionDataProvider

- ClientCollectionDataProvider
- ClientItemDataProvider
- JobCollectionDataProvider
- JobItemDataProvider
- PolicyCollectionDataProvider
- ScriptColectionDataProvider
- UserCollectionDataProvider

Klase guztiek dute inplementatuta *getCollection* edo *getItem* funtzioa, azken finean, oinarrian antzeko funtzioa dutena. Jarraian bilduma baten itzuleraren inplementazioa.

```
1 public function getCollection(string $resourceClass, string
      $operationName = null, array $context = [])
2     {
3         $repository = $this->entityManager->getRepository('App:User');
4         $query = $repository->createQueryBuilder('c')->addOrderBy('c.id',
       'ASC');
5         $queryNameGenerator = new QueryNameGenerator();
6 
7         foreach ($this->collectionExtensions as $extension) {
8             $extension->applyToCollection($query, $queryNameGenerator,
      $resourceClass, $operationName);
9             if ($extension instanceof
      QueryResultCollectionExtensionInterface && $extension->supportsResult(
      $resourceClass,$operationName)) {
10                 return $extension->getResult($query, $resourceClass,
      $operationName);
11             }
12         }
13         $this->logger->debug(
14             'View users',
15             array(),
16             array('link' => $this->router->generateUrl('showUsers'))
17             );
18         $this->entityManager->flush();
19 
20         return $query->getQuery()->getResult();
21 }
```

Data Persister

Informazioa datu-basean gorde edo bertatik ezabatzeari dagokionez, hauek *ContextAwareDataPersisterInterface* interfazea inplementatu dute, eta bi funtzio nagusi dituzte: *save* entitatea gordetzeko eta *delete* ezabatzeko. Klase hau bezero eta lanentzat bakarrik inplementatu behar izan da: *JobDataPersister* eta *ClientDataPersister*. Jarraian bezeroen funtzio nagusiak:

```
1 public function persist($data, array $context = [])
2 {
3     try {
```

```
4         $this->clientService->save($data);
5         return $data;
6     } catch (Exception $e) {
7         throw new InvalidArgumentException($e->getMessage());
8     }
9
10 }
11
12 public function remove($data, array $context = [])
13 {
14     try{
15         $this->clientService->delete($data->getId());
16     } catch (PermissionException $e) {
17         throw new PermissionException($e->getMessage());
18     }catch (Exception $e) {
19         throw new APIException($e->getMessage());
20     }
21
22 }
```

Service klaseak, zerbitzuen kapsulaketarako

API-a garatzerako orduan, hau dagoeneko inplementatutako aplikazio baten gainean ari dela eraikitzen kontuan izan behar da; eta ondorioz, ez dela *legacy* kodeaz ahaztu behar, hau da, dagoeneko garatuta dagoen kodeaz. Hori dela eta, kodea ahalik eta gutxien errepikatzeko saiakera egin behar da, batzuetan *legacy* kodean aldaketak eginez API-arentzat berrerabilgarria izateko helburuarekin. Azken finean, ez du zentzurik helburu berdina duten funtzio bat inplementatzea web-aplikaziorako, eta bestea API-rako.

Hau horrela izan dadin, *service* klaseak definitu dira. Orain arte, bezero eta lanak *DefaultController* klase kontroladoretik kudeatzen ziren. Orain, eragiketa hauek bai *Elkarbackup*-en kontroladoretik bai API-tik atzitzeko aukera izateko, funtzio hauek kapsulatzeko zerbitzuak sortu dira. Bost zerbitzu definitu dira:

- **ClientService.** Zerbitzu honek bi funtzionalitate eskaintzen ditu: bezeroak datubasean gorde eta bertatik ezabatu. Jarraian bi funtzio nagusiak:

```
1
2 public function delete($id)
3 {
4     $this->assertPermission($id);
5     $repository = $this->em->getRepository('App:Client');
6     $client = $repository->find($id);
7     $queue = $this->em->getRepository('App:Queue')->findAll();
8     foreach ($queue as $item) {
9         if ($item->getJob()->getClient()->getId() == $id) {
10            $this->logger->err('Could not delete client %clientName
      %, it has jobs enqueued.', array(
11                '%clientName%' => $client->getName()
12            ), array(
13                'link' => $this->router->generateClientRoute($id)
14            ));
15            throw new Exception("Could not delete client, it has
      jobs enqueued.");
16        }
17    }
```

```
18
19      $this->em->remove($client);
20      $msg = new Message('DefaultController', 'TickCommand',
        json_encode(array(
21          'command' => "elkarbackup:delete_job_backups",
22          'client' => (int) $id
23      )));
24      $this->em->persist($msg);
25      $this->em->flush();
26      $this->logger->info('Client "%clientid%" deleted', array(
27          '%clientid%' => $id
28      ), array(
29          'link' => $this->router->generateClientRoute($id)
30      ));
31  }
32
33  public function save($client)
34  {
35      ...
36      $clientName = $client->getName();
37      $repository = $this->em->getRepository('App:Client');
38      $existingClient = $repository->findOneBy(['name' => $clientName
        ]);
39      if (null != $existingClient) {
40          if ($existingClient->getId() !== $client->getId()){
41              throw new Exception("Client name ".$clientName." already
         exists");
42          }
43      }
44      if ($client->getOwner() == null) {
45          $client->setOwner($this->security->getToken()
46              ->getUser());
47      }
48
49      if ($client->getMaxParallelJobs() < 1) {
50          throw new Exception('Max parallel jobs parameter should be
        positive integer');
51      }
52      $this->em->persist($client);
53      $this->em->flush();
54      $this->logger->info('Save client %clientid%', array(
55          '%clientid%' => $client->getId()
56      ), array(
57          'link' => $this->router->generateClientRoute($client->getId
        ())
58      ));
59  }
```

- **JobService.** Bezeroen zerbitzuaren moduan, lanak datu-basean gorde eta bertatik ezabatzeko funtzioak ditu.

```
1  public function delete($id)
2  {
3      $job = $this->em->getRepository('App:Job')->find($id);
4      $idClient = $job->getClient()->getId();
5      $this->assertPermission($idClient);
6      $queue = $this->em->getRepository('App:Queue')->findAll();
7      foreach ($queue as $item) {
```

```
8           if ($item->getJob()->getId() == $id) {
9               $this->logger->err('Could not delete job %jobName%, it
    is enqueued.', array(
10                  '%jobName%' => $job->getName()
11              ), array(
12                  'link' => $this->router->generateJobRoute($id,
    $idClient)
13              ));
14              throw new Exception(sprintf('Could not delete job %s, it
     is enqueued.', $job->getName()));
15          }
16      }
17      $this->em->remove($job);
18      $msg = new Message('DefaultController', 'TickCommand',
    json_encode(array(
19          'command' => "elkarbackup:delete_job_backups",
20          'client' => (int) $idClient,
21          'job' => (int) $id
22      )));
23      $this->em->persist($msg);
24      $this->em->flush();
25      $this->logger->info('Client %clientid%, job "%jobid%" deleted
    successfully.', array(
26          '%clientid%' => $idClient,
27          '%jobid%' => $id
28      ), array(
29          'link' => $this->router->generateJobRoute($id, $idClient)
30      ));
31  }
32
33  public function save($job)
34  {
35      $this->em->persist($job);
36      $this->em->flush();
37      $this->logger->info('Save client %clientid%, job %jobid%', array
    (
38          '%clientid%' => $job->getClient()
39              ->getId(),
40          '%jobid%' => $job->getId()
41      ), array(
42          'link' => $this->router->generateJobRoute($job->getId(),
    $job->getClient()
43              ->getId())
44      ));
45  }
```

- **LoggerService.** *Log* mezuak gorde eta erregistratzen dituen zerbitzua. Hainbat maila bereizten dira, mezuaren edukiaren arabera: *debug*(jarraian), *info*, *warn* eta *err*.

```
1 public function debug($msg, $translatorParams = array(), $context =
    array())
2 {
3     $context = array_merge(array('source' => 'DefaultController'),
    $context);
4     $this->logger->debug(
5         $this->translator->trans($msg, $translatorParams, '
    BinovoElkarBackup'),
6         $context
```

```
7         );
8 }
```

6.1 Listing: LoggerService-ko funtzio baten adibidea

- **RouterService.** Helbide edo URI-ak itzultzeko zerbitzua. Bezeroak, lanak, politikak, script-ak, erabiltzaileak eta segurtasun-kopien kokalekuak editatzeko operazioen URI-a itzultzeko funtzioak ditu inplementatuta, baita beste helbideetarako funtzionalitate orokor bat.

```
1 public function generateJobRoute($idJob, $idClient)
2 {
3     return $this->router->generate('editJob', array(
4         'idClient' => $idClient,
5         'idJob' => $idJob
6     ));
7 }
```

- **TranslatorService.** Mezuen itzulpenerako *TranslatorInterface* zerbitzua erabiltzen du.

```
1 public function trans($msg, $params = array(), $domain = '
    BinovoElkarBackup')
2 {
3     return $this->translator->trans($msg, $params, $domain);
4 }
```

Dependentzia txertatzea (*Dependency injection*)

Garapenean zehar hainbat aldiz azaldu dira zerbitzuak, eta zerbitzu batetik beste hainbaten atzipen eta erabilpena. Horregatik, interesgarria da dependentzia txertatzea edo *dependency injection* kontzeptua azaltzea, funtzionamendua erakutsi eta adibideren bat aurkeztea.

Dependentzia txertatzea[4] software ingeniaritzako teknika bat da, non objektu bati beste objektu batzuk pasatzen zaizkion, honek sortu beharrean. Beste objektu horiei zerbitzu deituko zaie, eta jasotzen dituen objektua zerbitzu hauen menpeko izango da. Diseinu-patroi hau jarraituz, objektu bat sortu eta erabiltzearen ardurak banatu egiten dira, irakurerraztasuna eta kodea berrerabiltzeko aukerak areagotuz.

Hainbat txertaketa mota daude. Ezagunenak eraikitzailean, *setter*-etan eta interfazeetan ematen dira, eta aipatutako metodo hauen bidez txertatzen dira beharrezko zerbitzuak. Gradu Amaierako Lan honetan eraikitzailean txertatu dira zerbitzuak gehien bat. Hurrengo kode zatian ikus daiteke *ClientCollectionDataProvider* klaseko eraikitzailea, non hainbat zerbitzu ezberdin txertatu diren. Hala nola, datu-basea kudeatzeko *EntityManagerInterface* zerbitzua, erabiltzaileen baimenak kontsultatzeko *AuthorizationCheckerInterface*, kautotze informazioa atzitzeko *Security* eta *Service* klaseak, zerbitzuen kapsulaketarako atalean azaldutako *LoggerService* eta *RouterService* zerbitzuak.

```
1 private $authChecker;
2 private $collectionExtensions;
3 private $entityManager;
4 private $logger;
5 private $router;
6 private $security;
```

```
7
8  /**
9   * Constructor
10  */
11 public function __construct(EntityManagerInterface $em,
       AuthorizationCheckerInterface $authChecker, Security $security,
       LoggerService $logger, RouterService $router, iterable
       $collectionExtensions)
12 {
13     $this->authChecker          = $authChecker;
14     $this->collectionExtensions = $collectionExtensions;
15     $this->entityManager        = $em;
16     $this->logger               = $logger;
17     $this->router               = $router;
18     $this->security             = $security;
19 }
```

Emandako aldaketa eta arazoak

Hasierako diseinuan aurreikusitako ezaugarri eta funtzionalitateez gain, proiektua garatzen joan ahala aurkitutako behar eta gabeziak betetzeko aldaketa batzuk egin eta funtzionalitate batzuk gehitu edo kendu behar izan dira proiektuaren hasieran zehaztutako diseinutik (5 atala). Atal honetan azalduko dira puntuz-puntu egin diren moldaketa horiek.

Erabitzaileen baimenak

Proiektu honen hasieran, *Elkarbackup* aplikazioko erabiltzaileek bi baimen mota izateko aukera zuten: administrariak edo erabiltzaile soilak. Hau horrela izanda, API-aren funtzionalitateak diseinatzerako orduan, 5 atalean ikus daitekeen moduan (5.5 irudiko erabilpen kasuetan), baldintza hau kontuan hartu zen.

Hala ere, *Elkarbackup* bertsio berriak garatzen eta argitaratzen doaz, eta 2.0.0 bertsioan erabiltzaileen rolak zaharkitutzat eman eta hauen login-a blokeatu da; beraz, v2.0.0 bertsiotik aurrera erabiltzaile guztiek administrari baimena izan beharko dute. Horregatik, ez du zentzurik API-arentzat baimenen tratamenduarekin jarraitzea. Hau horrela izanda, erabiltzaileen baimenak oraindik guztiz ezabatuak ez daudenez eta API-an tratamendu hori garatuta zegoenez, kodea bere horretan utzi da, lan gehigarria ez suposatzeko. Hala ere, testetan ez dira txertatu baimenen tratamendua egiaztatzeko probak.

Menpeko baliabideak

Elkarbackup API-a diseinatzerako orduan lanak bezeroen menpeko bezala planteatu ziren, hau da, lan bat atzitzeko URI-a *clients/id/jobs/id* izatea pentsatu zen. Zoritxarrez, API Platform-en aukera horrek erreferentziarako bakarrik balio du, ezin dira horren gainean GET ez diren operazioak burutu. Hori dela eta, helbideen diseinuan aldaketak egin dira, eta lanetan *client* atributuaren bidez adierazten da zenen lana den. Ondorioz, lanen operazioen xehetasunak 6.1 taulan zehaztutako moduan geratu dira.

Horrez gain, bezero zehatz baten lanak lortzeko aukera izateko, filtro bat gehitu zaio lanak zerrendatzeko funtzionalitateari, lanak bezeroka iragazteko aukera izateko.

```
1 protected function filterProperty(string $property, $value, QueryBuilder
      $queryBuilder, QueryNameGeneratorInterface $queryNameGenerator, string
       $resourceClass, string $operationName = null)
```

	HTTP metodoa	CRUD funtzioa	URI identifikatzailea	Azalpena
Lan	GET	Read	/api/jobs	*ClientId* bezeroaren lanak zerrendatu
	GET	Read	/api/jobs/{jobId}	*ClientId* bezeroaren *jobId* lana eskuratu
	POST	Write	/api/jobs	*ClientId* bezeroaren lan berria sortu
	PUT	Replace	/api/jobs/{jobId}	Zehaztutako Id-a duen bezeroaren *jobId* lana, emandako datu berriekin ordeztu
	DELETE	Delete	/api/jobs/{jobId}	*ClientId* bezeroaren *jobId* lana ezabatu

6.1 Taula: *Elkarbackup* API-ko lanen operazioen notazio eta definizioa

```
2 {
3     if ($property === 'client'){
4         $parameterName = $queryNameGenerator->generateParameterName(
    $property);
5         $queryBuilder->andWhere(sprintf('j.%s = :%s', $property,
    $parameterName));
6         $queryBuilder->setParameter($parameterName, $value);
7     }
8 }
```

Baliabide gehigarriak

Irismenean aipatu moduan, orduen kudeaketaren arabera, interesgarria izan zitekeen bezero eta lanetatik erreferentziatzen ziren entitateak ere baliabide moduan gehitzea API-an, hauek zerrendatzeko aukera izateko. Nahiz eta ondoren testak oinarrizko funtzionalitateetan zentratu diren eta komando-lerroko interfazean ere ez diren dagozkien komandoak inplementatu, denbora kontuagatik batik bat, politika, script, erabiltzaile eta segurtasun-kopien lokalizazioak zerrendatzeko funtzionalitateak inplementatu dira API-an.

Horretarako, hauek izan dira garatu behar izan diren klaseak:

- **DTO**-ak. Baliabide hauen GET operazioan bakarrik inplementatuko direnez, irteerakoak inplementatu behar izan dira soilik.
  - *ScriptOutput*
  - *PolicyOutput*
  - *BackupLocationOutput*
  - *UserOutput*
- Irteerako DTO bakoitzerako ***data transformer***-a, datu-basetik lortuko diren entitateetatik irteerako DTO-etara bihurketa egiteko.
- Baliabide bakoitzaren ***data provider***-a, baliabide multzoak itzultzeko.

Filtroak

Behin oinarrizko funtzionalitate batzuk garatuta API-a proba moduan erabiltzen hastean, baliabideak zerrendatzean izenaren araberako filtroak gehitzeko beharra ikusi da, bai bezeroentzat bai lanentzat. Lanetan dagoeneko bezeroaren araberako filtroa garatu da, aurretik azaldu den moduan, beraz ez du lan-karga handia proposatu.

Horretarako kasu bakoitzerako filtro bat inplementatu behar izan da eta entitatean bertan definitu. Jarraian kode zati esanguratsuak. *filterProperty* funtzioan zehazten da filtroaren funtzionamendua; txertatutako kodeak ikusi daitekeenez, metodoari datu-baseari bidaliko zaion *query*-a zabaltzen du, *where* baldintza bat gehituz, eskatzen den izena duen bezero edo lana itzul dezan.

```
protected function filterProperty(string $property, $value, QueryBuilder
    $queryBuilder, QueryNameGeneratorInterface $queryNameGenerator, string
     $resourceClass, string $operationName = null)
{
    if ($property === 'name'){
        $parameterName = $queryNameGenerator->generateParameterName(
    $property);
        $queryBuilder->andWhere(sprintf('c.%s = :%s', $property,
    $parameterName));
        $queryBuilder->setParameter($parameterName, $value);
    }
}
```

Definitutako filtroa nahi den entitatean aplikatu ahal izateko, *@apiFilter* anotazioa gehitu beharko zaio dagokion atributuari, segidan ikus daitekeen modura.

```
protected $jobs;

/**
 * @ApiFilter(ClientByNameFilter::class)
 * @ORM\Column(type="string", length=255, unique=true)
 */
protected $name;
```

6.2 *Elkarbackup* API testing

Elkarbackup API-aren testen garapena bi ataletan bereizi daiteke: proben klase nagusiak eta fixtureen garapena. Jarraian azalduko dira atal bakoitzaren xehetasunak eta garapen prozesuan aurkitutako zailtasunak.

6.2.1 Proben inplementazioa

Proba klaseek *ApiTestCase* klasea zabaldu behar dute, eta proiektu honetan bi klase bereiztu dira: *ClientTest*, bezeroen funtzionalitateak probatzeko; eta *JobTest*, lanen funtzionalitateak egiaztatzeko. Dena den, kode komuna bilduko duen oinarrizko klase bat ere garatu da, aurrerago azalduko dena.

Horrez gain, datuak sortzeko *objectMother* klase motak erabili dira. Izan ere, nahiz eta aurretik aipatu fixtureak erabiliko direla datu-basea elikatu eta interesatzen den informazioa sortzeko, hainbat kasutan fixture bidez sortutako entitateak erabiltzea ez da komenigarria, horrek testen printzipioak(5.2.2) betetzea eragotziko luke eta, errepikagarritasuna hautsiz.

Adibide bat jartzearren, bezeroak ezabatzeko funtzionalitatea probatzeko testak inplementatzean fixture bidez sortutako bezeroak ezabatzen baditugu, test horiek berriro exekutatzean (datu-basea berrezarri gabe, noski) test horiek ez dute esperotako emaitza jasoko. Kasu horretan, komenigarria da testean bertan bezero bat sortu eta hori bera ezabatzea. Hainbat testetan bezeroak sortu behar badira, hauek sortzeko metodo batzuk inplementatzea interesgarria da, eta eginkizun hori du aurrerago azalduko diren *objectMother* klaseak.

Azkenik, probetan egiten diren operazioen datu eta erantzunak modu errazago batean aztertzeko aukera izateko, informazio hau *RequestObject* izeneko klaseen bidez egituratu da.

BaseApiTestCase

Kodearen errepikapenak saihesteko, ***BaseApiTestCase*** oinarrizko klasea garatu da, hainbat test-kasuetan erabiltzeko funtzioak biltzen dituena. Jarraian azalduko dira inplementatutako funtzioak.

- **assertHydraError**

  Operazioak itzulitako JSON-ean errore mezua itzuli duela egiaztatzen du, mezu lehenetsiarekin edo parametro bezala pasatako mezuarekin.

- **authenticate**

  HTTP bezeroa kautotzeko funtzioa. Ezaguna den *root* erabiltzailearekin egiten da autentifikazioa.

- **getScriptId**

  Scriptaren izena emanda, GET operazio baten bidez script-a jaso eta honen identifikazio zenbakia itzultzen du.

- **postClient**

  RequestObject-a pasata bezero bat sortzen duen funtzioa. HTTP eskaerak itzultzen duen informazioarekin RequestObject-a itzultzen du, *iri* eta *id* atributuak ezarrita.

- **postJob**

  RequestObject-a pasata lan bat sortzen duen funtzioa. *postClient*-en moduan, *id* eta *iri* atributuak definitzen ditu.

ObjectMother

ObjectMother[19] klaseak testetarako objektuak sortzeko lagungarri diren klase motak dira. Probetan objektu asko behar izaten dira, eta askotan ez dute garrantzirik objektuen atributuen balioek. Esaterako, bezeroen funtzionalitatea probatzeko askotan nahikoa da existitzen ez den izen bat duen bezero bat sortzea, gainontzeko atributuek ez dute garrantzirik.

Klase mota hauek, objektu hauek modu trinko batean sortzea ahalbidetzen dute, kodea errepikatzea saihestuz. Gainera, kodearen semantika zaintzen du, egitura ulerterraza emanez. Hemen, existitzen ez den jabea emanez bezeroa sortzean izango duen jokaera probatuko duen testean bezeroa sortzeko deia:

```
$client = ClientMother::withNonExistentOwner();
```

Horregatik, bi klase inplementatu dira; bat bezeroentzat eta bestea lanentzat, eta jarraian azalduko dira klase bakoitzak sortzen dituen objektu motak.

- ClientMother

  Bezeroen *RequestObject*-ak itzultzen dituzten metodo estatikoak biltzen dituen klasea. Hauen artean izango dira oinarrizko parametroak balio lehenetsiekin itzuliko duena (*base*), parametro guztiak pasatzekoak(*withAllParameters*), *maxParallelJobs* atributu okerra izango duena(*withInvalidMaxParallelJobs*), existituko ez den jabea izango duena (*withNonExistentOwner*), existitzen ez den *postScript* bat izango duena (*withNonExistentPostScript*)... besteak beste.

- JobMother

  Lanen *RequestObject*-ak itzultzen dituzten metodo estatikoak biltzen dituen klasea. Honetan ere, oinarrizko metodo estatiko bat izango du, objektua lehenetsitako beharrezko balioekin itzuliko duena; beste klase bat parametro guztiak zehazteko, eta ondoren frogatu beharreko atributuak banaka zehaztu edo zuzenean atributu okerrarekin itzuliko duena.

RequestObject

Operazio bakoitzean bidali eta jasotzen den informazioa atzitu eta egiaztapenak egiteko, *RequestObject* izeneko klasea inplementatu da. Klase honek hainbat atributu izango ditu: *context*, *data*, *id* eta *iri*. Objektu berri bat sortzean, dagokion *objectMother* klasearen bidez, *Context* atributuak entitatearen ezaugarriak gordetzen ditu:

```
const CLIENT_CONTEXT = [
'@context' => [
    '@vocab' => 'http://127.0.0.1/api/docs.jsonld#',
    'hydra' => 'http://www.w3.org/ns/hydra/core#',
    'description' => 'ClientOutput/description',
    'id' => 'ClientOutput/id',
    'isActive' => 'ClientOutput/isActive',
    'maxParallelJobs' => 'ClientOutput/maxParallelJobs',
    'name' => 'ClientOutput/name',
    'owner' => 'ClientOutput/owner',
    'postScripts' => 'ClientOutput/postScripts',
    'preScripts' => 'ClientOutput/preScripts',
    'quota' => 'ClientOutput/quota',
    'rsyncLongArgs' => 'ClientOutput/rsyncLongArgs',
    'rsyncShortArgs' => 'ClientOutput/rsyncShortArgs',
    'sshArgs' => 'ClientOutput/sshArgs',
    'url' => 'ClientOutput/url'
],
'@type' => 'Client'
];
```

data atributuan objektuaren balioak gordeko dira. Objektu horren POST egitean berriz, *BaseApiTestCase* klaseko *postClient* edo *postJob* funtzioen bidez, *iri* eta *id* atributuak ezarriko dira, aurrerago objektu hauek atzitu ahal izateko.

```
protected function postJob(Client $httpClient, RequestObject $job):
    RequestObject
{
    $jobJson = $job->getData();
    $response = $httpClient->request('POST', '/api/jobs', [
        'json' => $jobJson
    ]);
    if (201 == $response->getStatusCode()){
        $json = json_decode($response->getContent(), true);
        $job->setIri($json['@id']);
        $job->setId(explode('/', $json['@id'])[3]);
    }
    return $job;
}
```

6.2.2 Fixtures

Fixtureak datu-basea elikatzeko erabiliko dira; batik bat, bertatik lortu bakarrik nahiko direnak, hau da, GET operazioetarako bakarrik, testen errepikakortasuna ez hausteko. Horretarako, batez ere script ezberdinak izatea interesgarria da, API-aren bidez ezingo baitira sortu eta probetarako mota ezberdinetakoak eduki beharko dira. Hala ere, bezero eta lanenbat izatea komenigarria izango da probaren bat egiteko.

Scripts

Esan bezala, script-en tratamendua probatzeko hainbat script mota ezberdin izango dira. Horregatik, batetik kasu berezi bakoitzerako script bat sortu da, eta bestetik hainbat script hausazko edukiarekin.

```
script_not_job_post:
      __construct: ['/tmp/elkarbackup-tests/uploads']
      description: <text()>
      name: 'script_not_job_post'
      scriptFile: <upload('fixtures/script.sh')>
      isClientPre: <boolean()>
      isJobPre: <boolean()>
      isClientPost: <boolean()>
      isJobPost: false
```

Ikusi daitekeenez, *upload* funtzioa erabiltzen da *UploadedFile* motako atributua txertatzeko. Hurrengo atalean (6.2.2) azalduko dira metodo honen nondik norakoak. Parametro bezala pasatzen zaion *script.sh* fitxategiari dagokionez, eduki esanguratsurik ez duen script fitxategi bat da, *UploadedFile* motako atributua sortzeko aukera izateko.

Zailtasunak

Aipatu bezala, script fixtureak sortzeko, *scriptFile* atributuan scriptak UploadedFile motako objektu bat simulatzeko modua bilatu behar izan da. Soluzio moduan, *faker* klase bat inplementatu da, *upload* funtzioarekin. Honek fitxategi baten (*script.sh*) helbidea (path?) jasota, fitxategi hori igo eta honen *UploadedFile* motako objektua itzuliko du.

```
<?php

class ScriptFileFaker extends \Faker\Provider\Base
```

```
{
    public function upload($filename)
    {
        if (is_array($filename)) {
            $filename = \Faker\Provider\Base::randomElement($filename);
        }

        $path = sprintf('/tmp/elkarbackup-tests/uploads/%s', uniqid());

        $copy = copy($filename, $path);

        if (! $copy) {
            throw new \Exception('Copy failed');
        }

        $mimetype = MimeTypeGuesser::getInstance()->guess($path);
        $size = filesize($path);

        $file = new UploadedFile($path, $filename, $mimetype, $size, null
    , true);

        return $file;
    }
}
```

6.3 *Elkarbackup* API CLI

Gradu Amaierako Lan honen azken garapen atala API-a kontsumitzen duen komandolerroko bezeroari dagokio. Hau *Symfony*-ren kontsola osagaia erabiliz inplementatu da. Bi multzotan sailka daiteke; kontsola bera, eta bertan txertatutako komandoak.

Kontsola

Kontsola PHP exekutagarri batean definitzen da. Bertan *Symfony*-k eskaintzen duen *Application* objektuaren bidez gehitzen dira definitutako komandoak. Behin nahi diren komandoak gehituta, aplikazioa martxan (*run*) jarriko da.

```
$application = new Application('elkarbackup-api-cli', '@package_version@'
    );
$application->add(new Command\DeleteClientCommand());
$application->add(new Command\GetClientsCommand());
...
$application->add(new Command\UpdateJobManualCommand());
$application->add(new Command\UpdateJobFromFileCommand());
$application->run();
```

Komandoak

Komando bat inplementatzeko *Command* klasea zabaltzen duen klase bat garatu behar da. Diseinuan zehaztu moduan, komando guztiek parametro komun batzuk izango dituzte (erabiltzailea, pasahitza eta url-a konfiguratzeko aukera), hauek komando bakoitzean definitu behar ez izateko, ***BaseCommand*** klasea definitu da. Era berean, hainbat funtzio laguntzaile definitu dira bertan.

BaseCommand klaseko funtzio esanguratsuenak aipatzearren, batetik ***manageError*** metodoak operazio baten erantzuna jasota, honek itzulitako errore kodearen arabera, mezua inprimatu eta dagokion kontsolako kodea itzuliko du.

```
switch ($status) {
    case 401:
        $output->writeln($message);
        return self::UNAUTHORIZED;
    case 404:
        $output->writeln($message);
        return self::NOT_FOUND;
    case 422:
        $output->writeln($message);
        return self::INVALID_ARGUMENT;
    default:
        $output->writeln($message);
        return self::ERROR;
}
```

Ikus daitekeen bezala, *manageError* funtzioak errore-kodeak tratatuko ditu. Eragiketa arrakastatsua bada komandoan bertan tratatuko da, ez zaio metodo honi dei egingo. Dena den, hauek dira definitu diren itzulera-kodea guztiak:

```
    const SUCCESS = 0;
    const ERROR = 1;
    const UNAUTHORIZED = 2;
    const INVALID_ARGUMENT = 3;
    const NOT_FOUND = 4;
    const COMMUNICATION_ERROR = 5;
```

Beste funtzio esanguratsua, parametro komunak gehituko dituen ***configure*** izango da. Hau *Command* klaseak definitutako metodo bat da, eta bertan zehazten dira komandoaren izena, deskribapena eta izango dituen argumentu eta aukerak.

```
protected function configure(): void
{
    $this
        ->addArgument('username', InputArgument::REQUIRED, "Username for
    authentication")
        ->addArgument('password', InputArgument::REQUIRED, "Password for
    authentication")
        ->addOption('apiUrl', null, InputOption::VALUE_OPTIONAL, "Url of
    the api", "http://127.0.0.1")
    ;
}
```

Komando bakoitzari dagokionez, denek *BaseCommand* klasea zabalduko dute, eta bi metodo izango dituzte: *configure*, *BaseCommand*-en definitutakoa zabalduko duena; eta *execute*, komandoa exekutatzean egingo duenaz arduratuko dena.

Adibide moduan, bezero baten xehetasunak lortzeko komandoa aztertuko da, hau da, *GetClientCommand*. Lehenik eta behin, hau da komandoaren konfigurazioa:

```
parent::configure();
$this
    ->setName('client:details')
    ->setDescription('Get a client')
    ->addArgument('id', InputArgument::REQUIRED, "Client's id")
;
```

Oinarrizko argumentuez gain, lortu nahi den bezeroaren identifikazio zenbakia pasa beharko da. Komandoa exekutatu eta argumentuen balioak jasotzean, lehen pausoa *id* argumentua integer motara bihurtzea izango da. Horretarako, *BaseCommand*-eko *parseInt* funtzioari egingo zaio dei. Pasatako balioa ez bada zenbaki bat, salbuespen bat jaurtiko du eta komandoan salbuespen hau arrapatu eta behar den mezua inprimatuko du, kode egokia itzultzearekin batera.

```
try {
    $id = $this->parseInt($input->getArgument('id'));
} catch (\InvalidArgumentException $e) {
    $output->writeln("Id of the client must be a integer");
    return self::INVALID_ARGUMENT;
}
```

Behin egiaztapen hau eginda, HTTP GET eragiketa egingo da, URI-an beharrezko parametroak txertatuz.

```
$response = $httpClient->request('GET', $url.'/api/clients/'.$id, ['
    auth_basic' => [$username, $password],]);
```

Ondoren, *response* aldagaiak jasotako erantzuna aztertu beharko da. Lehenik eta behin, itzuli duen egoera-kodea begiratuko da. Horrek salbuespen bat jaurti dezake, beraz honen tratamendua egin beharko da.

```
try {
    $status = $response->getStatusCode();
} catch (TransportException $e) {
    $output->writeln($e->getMessage());
    return self::COMMUNICATION_ERROR;
}
```

Jasotako kodea arrakastatsua bada, hau da, kasu honetan 200 kodea jaso bada, zuzenean jasotako bezeroaren xehetasunak inprimituko dira kontsolan eta arrakasta-kodea itzuliko da. Bestela, *manageError* funtzioari deituko zaio errorearen tratamendua egin dezan.

```
if (200 == $status) {
    $output->writeln($response->getContent());
    return self::SUCCESS;
}
return $this->manageError($response, $output);
```

Hau izango litzateke bezeroen xehetasunak lortzeko komandoak ematen dituen pausoak. Komando guztiek egitura oso antzekoa dute, gehien bat dituen parametroen arabera ematen zaien tratamendua aldatuko litzateke. Horren harira, aipatu objektuen parametro guztiak txertatzeko aukera dagoen komandoetan, hau da, bezero eta lanak parametroak eskuz sartuta sortu edo ordezten denetan, parametroen irakurerraztasuna bermatzeko *option* bezala definitu direla parametro hauek. Ondorioz, definizioz ez dira derrigorrezkoak, eta kontsolak defektuz ez du arazorik emango parametro hauek jartzen ez badira. Kasu honetan, bezero eta lanen atributu batzuk derrigorrezko direnez, *checkRequiredOptionsAreNotEmpty* funtzioa inplementatu da *BaseCommand* klasean, derrigorrezko diren atributu hauek ez badira zehazten salbuespena jaurti eta errore-tratamendua egin dadin.

```
$options = $this->getDefinition()->getOptions();
foreach ($options as $option) {
    $name  = $option->getName();
    $value = $input->getOption($name);
```

```
5     if ($option->isValueRequired() && (null == $value || '' == $value)) {
6         throw new \InvalidArgumentException(sprintf('The required option
      %s is not set', $name));
7     }
8 }
```

7

Jarraipena eta kontrola

Kapitulu honek bilduko ditu Gradu Amaierako Lan honetako jarraipena burutzeko eraman den metodologia; hasiera batean zehaztutako plangintzarekiko izan diren desbiderapenak, bai ordu kopuruari dagokionez, baita Gantt diagraman definitutako epeekiko; kudeaketa atalean aipatutako arriskuetatik eman direnen azalpena; eta azkenik, amaierako proiektuaren kalitatea eta proiektuan zehar bertan parte hartu dutenekin ezarritako komunikazioa.

7.1 Erabilitako jarraipen metodologia

Proiektuaren jarraipena kalkulu-orrien fitxategi baten bidez burutu da batez ere, B eranskinean aurkitu daiteke dokumentu hau. Bertan bi orri edo taula bereizi dira, bat ordu-kopuruen jarraipena eramateko, eta bestea atazak burutzeko epeak erregistratzeko.

Ordu-kopuruen taula honela antolatzen da: zutabeka proiektuaren atazak bereizten dira, eta errenkada bakoitza egun bat izango da; lan-egun bakoitzean adieraziz ataza bakoitzari zenbat denbora dedikatu zaion. Horrela, alde batetik, egun bakoitzean dedikatu den ordu-kopuru osoa kalkulatuko da; eta bestetik, proiektuan aurrera egin ahala, ataza bakoitzari esleitutako orduak gehituko dira, baita proiektu osorako erabilitako ordu-kopurua. Gainera, hasierako plangintzan ezarritako estimazioekin konparaketa egiten da.

Horrez gain, atazen epeak gordetzeko bigarren taulan, errenkada bakoitza Gradu Amaierako Lanaren ataza bati dagokio, eta zutabeak egunka banatuta daude. Lan-egun bakoitzean landutako ataza koloreztatu da, Gantt diagrama moduko bat osatuz. Taula honi esker, amaierako Gantt diagrama (7.1) osatzeko erraztasunak izan dira, proiektua garatu ahala osatzen joan baita.

7.2 Plangintzarekiko desbiderapenak

Gradu Amaierako Lan honi hasiera ematean, lehen pausoetako bat plangintza osatzea izan zen. Bertan, proiektua aurrera eramateko egin beharreko lan eta atazak zehaztu eta izan zitzakeen arriskuak kontuan harturik, ordu eta epeen estimazioa finkatu zen (ikus 2.1 taula eta 2.2 diagrama).

Horrelako proiektu batean estimazio zehatzak egitea ez denez erreza, kontuan hartu da desbiderapenak egon zitezkeela, baina ezinbestekoa da ahalik eta estimazio zehatzena egitea proiektua arrakastaz burutu ahal izateko.

Ataza	Estimazioa (h)	Erreala (h)	Desbiderapena (h)
Ikasketa	20	24	4
Rest API	5	4	-1
API-platform	15	20	5
Garapena	180	167	-13
Funtzionalitate nagusiak	80	71	-9
Test automatikoak	50	62	12
Komandoko bezeroa	50	34	-16
Diseinua	10	12	2
Kudeaketa	50	38,5	-11,5
Jarraipena eta kontrola	20	19,5	-0,5
Plangintza	15	8	-7
Bilerak	15	11	-4
Dokumentazioa	70	76	6
Memoria	70	76	6
~~Defentsa~~	0	0	0
Denera	330	317,5	-12,5

7.1 Taula: Proiektuaren ataza bakoitzari dedikatutako ordu-kopuru erreala

Jarraian alderatuko dira (7.1 taulan) hasierako estimazioak eta proiektuaren amaierak izandako ordu-kopuru errealak. Horrekin batera, hasieran estimatutako eta amaierako Gantt diagramen (ikus 2.2 irudia) arteko ezberdintasunak ere azalduko dira. Azterketa honekin, proiektuaren amaierako balorazio orokor bat egin eta ondorioak ateratzeko ikuspegi bat eskainiko da.

Orduen desbiderapenak

Aurretik aipatu moduan, atal honetan proiektuaren hasieran estimatu zen ordu-kopuruak eta azkenean benetan izan direnen artean alderaketa egingo da, izandako desbiderapena aztertuko da. 7.1 taulan ikusi daitekeenez, plangintza osatzerakoan eraikitako 2.1 taulari bi zutabe gehitu zaizkio; bat ataza bakoitzari eskainitako ordu kopuru errealekin, eta bestea eman den desbiderapenarekin. Aipatu amaierako orduetan defentsan inbertitutako orduak ez direla kontuan izango, hau behin memoria amaituta burutuko baita.

Taulan azaltzen diren desbiderapenei erreparatzen badiegu, garapenean izan da desbiderapen esanguratsuena. API-aren funtzionalitate nagusiak garatzerako orduan estimatutako baino ordu gutxiago eman dira, baina ezberdintasun handirik gabe. Testen garapenean berriz, desbiderapena handiagoa izan da, 13 orduko aldea izanik.

Dena den, aipatu testen garapenean orduetan baino esanguratsuagoa izan dela desbiderapena epeetan, hurrengo atalean azalduko den bezala. Horrek azken garapen atalerako denbora mugatua izatea eragin du, eta ondorioz komando-lerroko bezeroaren atala garatzeko estimatutakoa baina ordu gutxiago izan dira.

Horrez gain, ez da desbiderapen esanguratsurik izan, eta Gradu Amaierako Lana estimatutako baina ordu gutxiagotan arrakastaz aurrera eramatea lortu da. Ikus daitekeenez, proiektua 330 orduetan egitea estimatu zen (defentsaren 20 orduak kontuan izan gabe), eta azkenean 318 ordu behar izan dira, hau da, desbiderapena %4-koa izan da.

Amaierako Gantt diagrama

Atal honetan berriz, amaierako Gantt diagrama aurkeztuko da, proiektuak izan dituen epe errealak azaltzeko eta plangintza diseinatzerakoan egin zen estimazioarekin alderatzeko.

Lehenik eta behin, proiektuaren hasieran 2021eko martxoaren 15ean ezarri zen. Dena den, enpresarekin proiektuari hasiera emateko bilera martxoaren 18an egin zen, eta ikasketa fasea hurrengo astean, hau da, martxoaren 22an hasi zen. Honen arrazoia ulertzeko, aipatu egilea Gradu Amaierako Lanarekin hasi aurretik enpresa berean ibili dela hautazko praktikak burutzen, eta bertan *Elkarbackup* aplikazioari eguneraketa, zehazki Symfony ingurunearen *upgradea*, eta zuzenketak egiten ibili dela, gerora Gradu Amaierako Lana garatu ahal izateko. Hau horrela, proiektuaren lehen astea izan behar zuen hori praktiketako eginkizunei azken ukituak egiteko erabili zen.

Aste beteko atzerapen horren ondorioz, plangintza osoa estimatutakoa baina beranduago burutu da. Hala ere, arriskuen kudeaketa egoki batekin, ez du eragin larririk izan.

Bestalde, API-ko bezeroen funtzionalitateak garatzeko estimatu baina epe zabalagoa behar izan da; dena den, autentifikazioa pare bat egunetan garatu denez, oreka mantentzeko aukera egon da.

Azken puntu bezala, eta agian eragin handiena izan duena, proben garapenarekin behin API-aren garapena amaituta hasi zen; horrek bai, desbiderapena suposatu du plangintzan, testak espero baina beranduago amaituz, eta komando-lerroko bezeroaren inplementaziorako denbora mugatua utziz amaierako data, maiatzak 21a, baino lehen. Hala ere, orduen estimazioan ere aipatu den moduan, CLI-aren inplementazioan lan egiteko nahikoa ordu izan dira, eta proiektua arrakastaz burutzeko gai izan da.

7.3 Arriskuak

Esan daiteke proiektuaren garapena nahiko konstantea izan dela eta ez dela lan-erritmoa oztopatu duen ustekabe larririk egon. Hala ere, kudeaketa planean zerrendatu ziren arrisku posibleetatik, hurrengo hauek aipatu daitezke.

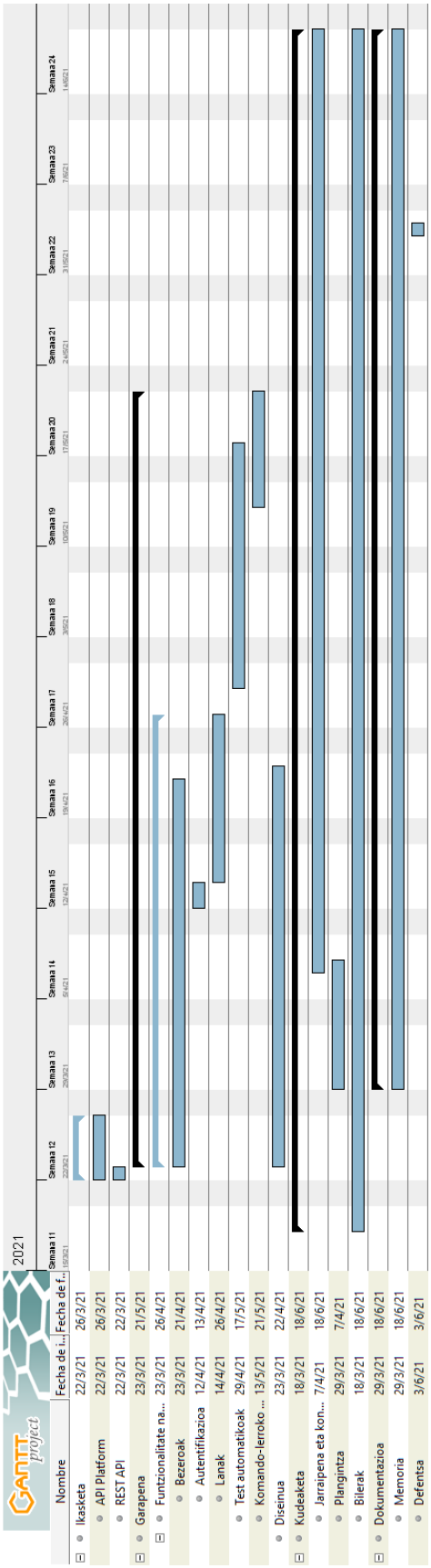
Osasun arazoak

Zorionez, proiektuaren egileak ez du lana egitea eragotzi dion osasun-arazorik izan. Hala ere, ezinbestekoa da aipatzea gaur egun Covid-19-arengatik ematen ari diren baldintza bereziak. Esaterako, egileak 8 egunez isolatuta egon behar izan du gertuko kontaktu bat positibo izanagatik. Horrek, proiektua garatu ahal izateko ordenagailua eskura ez izatea ekarri du, API-aren garapena etenez aste betez.

Dena den, esan behar da ez duela ondorio larririk suposatu; Aste Santuko oporretan izanik, kontuan hartu zen data bereziak zirela eta egun horietan lan-erritmoa jaitsi zitekeela. Gainera, nahiz eta aplikazioaren garapena eten, Gradu Amaierako Lanaren memorian aurrerapenak egiteko aukera egon da.

Estimazio okerrak

Aurreko atalean (7.2) aipatu bezala, hasierako plangintzan estimazio zuhurrak egitea ezinbestekoa da proiektua arrakastaz burutzeko; eta hasieratik kontuan hartzeko arrisku bat izan da zehaztutako estimazioak okerrak izatea. Hala ere, nahiz eta estimazioak ez diren guztiz zehatzak izan, eta kasuren batean desbiderapen esanguratsuren bat egon, esan behar da arazorik gabe kudeatu den arrisku bat izan dela, eta ez dela larritasunik egon proiektua arrakastaz bukatzerako orduan.



7.1 Irudia: Proiektuaren amaierako Gantt diagrama

Diseinu desegokia

Diseinuari dagokionez, esan daiteke nahiko modu zuhurrean aurreikusi dela proiektuak izango zuen egitura; hala ere, uneoro proiektuaren gaineko jarraipen eta kontrola eramanez, egin diren aldaketek ez dute arazo handirik suposatu.

Bat aipatzekotan, testen egitura ez zen nahikoa zehaztu agian; lehenengo bezeroen testak garatu ziren, eta hainbat moldaketa behar izan zituen, ez bakarrik helburu funtzionalak bete zitzaten, baizik eta arkiterktura zaindua ere izan zezaten. Adibidez, *objectMother* klaseak gerora txertatu ziren objektuak modu erraz eta trinkoago batean sortzeko aukera izateko. Hau argi eta garbi islatu daiteke denboraren desbiderapenetan, testen garapenean estimatu baina ordu gehiago eman baita, eta gantt diagraman aurreikusitako epea atzeratu egin da.

7.4 Kalitatea

Gradu Amaierako Lan honen garapen prozesu osoan zehar, honen kudeaketa plana egitetik garapena bukatu eta ondorioak ateratzeraino, ahalik eta kalitate maila altuena mantentzen saiatu da. Esan beharrik ez dago, *Binovo* enpresak proposatutako proiektua izanik, honek ezarritako kalitate-irizpideak bete behar izan direla. Horrez gain, kalitatearen inguruko hainbat puntu bereizi daitezke.

Batetik, **denboraren kalitatea** bermatzea gakoa izan da proiektua arrakastaz burutzeko. Dedikatu diren orduak ahalik eta gehien aprobetxatu nahi izan dira, distrazioak ekidinez alde batetik, baina baita produktibo mantentzeko beharrezko deskantsuak hartuz. Gainera, jarraipen metodologia (ikus 7.1 atala) egoki bat ere oso lagungarria izan da uneoro proiektuaren egoeraz jakitun izan eta denbora probesteko.

Bestetik, **kodeari** dagokionez, kudeaketa-planeko kalitate-irizpideetan zehaztu bezala, kode koherente eta zaindua garatu da, enpresaren estandarra (A eranskina) jarraituz eta baita garapenean erabili diren lan-inguruneen, hau da, *Symfony* eta *API Platform*-en praktika egokiak kontuan hartuz. Horrez gain, etorkizunean garatutako aplikazioarekin lanean ibili daitekeenarentzat ulergarritasuna bermatzeko, iruzkinak txertatu dira beharrezko ikusi den puntuetan.

Azkenik, **dokumentazioaren kalitateari** dagokionez, honen forma eta edukia zaindu da. Zuzentzailea erabili da akatsak ekiditeko, eta horrez gain, memoriako kapituluak garatzen joan ahala tutoreari bidali zaizkio zuzendu dezan. Behin memoria osatuta, enpresako tutore eta langile bati bidali zaio, berrikusi eta moldaketaren bat proposatzeko. Kodearekin bezala, dokumentazio txukun eta zuzen bat oso lagungarria izan daiteke etorkizunean aplikazioa landu nahi duenarentzat.

7.5 Komunikazioa

Proiektua etxetik garatua izan denez eta pandemia-egoeraren ondorioz izan diren murriztapenekin, komunikazio gehiena eposta bidez egin da. Enpresako tutoreekin *hangouts* txata eta *bigBlueButton* bideo-deiak ere erabili dira komunikazio puntual eta informalagoetarako, edozein zalantza argitzeko adibidez, edo baita bilerak egiteko.

Unibertsitateko tutorearekin berriz, bilera presentzial bat egin da proiektuaren hasieran, nondik norakoak azaldu eta zereginen xehetasunak azaltzeko. Gainontzeko komunikazioa

eposta bidez egin da, bai memoriaren zuzenketak bidaltzeko, bai edozein zalantza edo kontsulta egiteko.

8

Ondorioak

Proiektuaren garapena atalez atal azaldu ondoren, kapitulu honetan lortutako emaitza ebaluatu eta honen inguruko hausnarketa egingo da. Era berean, garatutako proiektuak etorkizunean izan ditzaken aukerak aztertu eta egileak ikasitako lezioak aurkeztuko dira.

8.1 Amaierako hausnarketa

Lehenik eta behin, esan orokorrean proiektua arrakastaz burutu dela, eta hasieran ezarritako helburuak bete direla. Baina aurrera egin aurretik, Gradu Amaierako Lan honetarako ezarritako helburuak errepasatuko dira.

Proiektu honetan, *Binovo* enpresaren *Elkarbackup* aplikazioa kudeatzen zuen RESTful API bat garatu behar zen, oinarrizko funtzionalitate batzuekin; hauek dira, bezeroak sortu, zerrendatu, ordeztu eta ezabatzea, eta lanak sortu, zerrendatu, ordeztu eta ezabatzea. Horrez gain, API-aren funtzionamendua frogatzeko test automatikoak garatu eta API-a komando lerrotik kontrolatzeko interfazearen garapena egitea zuen helburu Gradu Amaierako Lanak.

Esan daiteke zehaztutako helburu hauek bete direla, eta irismenean (ikus 2.3 atala) aipatutako funtzionalitate gehigarriak modu partzialean inplementatu direla. Hau da, API-ak politika, script, babes-kopien lokalizazio eta erabiltzaileak zerrendatzeko funtzionalitateak gehituta ditu, baina testen eta komando-lerroko bezeroaren zatia garatzeke geratu da, 6.1 atalean azaldu eta arrazoitzen den bezala.

Gainera, proiektuaren egileak burututako API-ak enpresan duen erabilera erreala ikusteko aukera izan du. Hasieran aipatutako (ikus 2.1 atala) ERP-en probisionamendu aplikazio bat garatzen ari dira, zerbitzuak modu automatikoan emateko. Aplikazio honetan, *Ansible*[1], IT automatizazio tresna, bidez inplementatutako probisionamendu-scriptekin osagai edo ataza ezberdinak ezarri eta konfiguratzen dira; esaterako, DNS sarrera sortzen da ezarri den azpidomeinu izenarekin, ERP aplikazioa instalatu, datu-basearen instantzia sortu... eta horien artean *Elkarbackup* aplikazioa instalatu eta konfiguratzen da datuen segurtasun-kopiak egikaritu eta kudeatzeko. Azken script honetan kontsumitzen da Gradu Amaierako Lan honetan garatutako API-a. Ondorioz, eskuz eginda gutxi gorabehera 2 ordu tardatuko zituen prozesu batek, 5-10 minututan egitea ahalbidetzen du automatizazioak.

[1] Ansible, https://www.ansible.com/

Horrez gain, API-a proiektu honetan inplementatutako komando-lerroko bezerotik kontsumitzeko aukeraz gain, *Binovo* enpresak *Ansible collection* bat, *Ansible*-ko script-en distribuzio formatu bat, ere garatu du honetarako. *Collection* hau hurrengo estekan aurki daiteke eskuragarri: https://galaxy.ansible.com/elkarbackup/elkarbackup

Plangintzari dagokionez, esan daiteke nahiko estimazio zuhurrak eta errealistak egin zirela, nahiz eta desbiderapenak izan eta egileak ezarritako atazen epeak ez diren guztiz bete. Hala ere, desbiderapenen kudeaketa egokia egin da, eta inplementazioa garaiz bukatzeko modua izan da.

Bukatzeko, amaierako produktuan jartzen bada arreta, argi dago hobekuntzak egin daitezkeela, funtzionalitateak osatuz batik bat, baina orokorrean enpresak ezarritako baldintza eta beharrak modu egokian asetzen ditu.

8.2 Etorkizunerako aukerak

Esan bezala, proiektu honek baditu hobekuntzak egiteke, eta esan daiteke hori dela aukera nagusia etorkizunera begira. Hau da, bezero eta lanez gain, beste entitate edo baliabideen kudeaketa edo garapena txertatzea test automatiko eta komando-lerroko interfazean.

Gainera, software libreko aplikazioa izanik, honen inguruko komunitateak aplikazioa probatzen eta erabiltzen joan ahala, hobekuntza eta moldaketak proposatzen joan daiteke.

Dena den, ez da ahaztu behar API-a dagoeneko existitzen zen aplikazio baten gainean dagoela eraikia, hau da, *Elkarbackup*-en garapenaren menpeko dela, eta horrek zabaltzeko aukera mugatuak izan ditzake, edo *Elkarbackup* aplikazioarekiko dependentzia izango du behintzat.

8.3 Ikasitako lezioak

Gradu Amaierako Lan hau enpresa batean burutzeko aukera izateak hainbat ikasketa ekarri ditu. Jarraian aipatuko dira puntuz puntu.

Lehenik eta behin, unibertsitateko proiektuetatik enpresa batekora jauzia egitean, alde batetik, lan-mundura gerturatu eta enpresa baten funtzionamendua ezagutzeko aukera izan da; eta bestetik, proiektu handiagoekin lan egiten eta hauek kudeatu eta garatzen ikasi da. Horretarako aurretiko plangintza eta diseinuak duen garrantzia azpimarratuz.

Horren harira, aurretik sortuta dauden proiektu bat aztertu, ulertu eta horren gainean lan egiten ikasi da; era berean, kode-estilo zaindu bat izateak duen garrantziaz jabetuz. Izan ere, proiektua ez du egileak soilik ulertu behar, etorkizuneko garatzaileentzat ere ulergarria izan behar du, proiektuak hobetuz joan daitezen.

Horrez gain, lan-ingurune berriak landu eta hauek erabiltzen ikasteko aukera izan da, batez ere *Symfony* eta *API Platform* tresnak, hainbat aukera zabal eta ezberdin eskaintzen dituztenak. Era berean, Software Ingeniaritzako hainbat printzipio edo diseinu-estilo ikasi eta barneratzeko aukera ere izan du egileak; hala nola, API-en funtzionamentua, REST printzipioak, ObjectMother klaseak, etab. Etorkizuneko beste proiektuetarako ere erabilgarriak izan daitezkeenak.

Eranskinak

A

Binovo-ko PHP kode estandarra

Binovo PHP Coding Standard v3

Fecha	**Cambio**
2015-07-14	Adecuar el coding standard anterior a los estándares PHP-FIG: PSR-0, PSR-1, PSR-2, PSR-4
2016-05-03	Corregir fecha de documento, publicarlo
2016-07-05	Luis detecta error en el ejemplo de Nombres de elementos de array, faltan llaves alrededor del elemento del array en el print
2018-06-19	ales – Añadir punto “Formato de bucles” y corregir el formato en un ejemplo de “Uso de *continue, break*”
2018-10-01	ales – Añadir punto “Uso de operadores” y clarificar punto “Longitud de línea”
2021-01-21	uanabitarte - Limitar el coding standard a las desviaciones y puntos añadidos al estandar PSR-12

1. Introducción

1.1. La estandarización es importante

El estándard nunca contenta a todos, siempre hay cosas que no nos van a gustar. Esta suele ser una señal bastante inequívoca de que es un estándard basado en la experiencia de mucha gente y no las preferencias de una persona.

1.1.1. Bondades de un estándar de programación

Cuando un proyecto trata de adherirse a un estándar común conseguimos lo siguiente:

- Los programadores podemos ir a cualquier código y adivinar qué es lo que está ocurriendo
- Los nuevos programadores son productivos más rápidamente
- Las personas que acaban de aprender PHP evitan cometer errores que muchos otros antes que ellos cometieron
- En un entorno consistente, las personas tendemos a cometer menos errores
- Los programadores tenemos un enemigo común :-)

1.1.2. Problemas

- El estándar es generalmente estúpido, porque no es lo que hago yo
- Los estándares reducen la creatividad
- Los estándares fuerzan demasiada estructura/adorno
- La gente ignora los estándares

1.1.3. Discusión

La experiencia de muchos proyectos concluye que usar un estándar de código hace que el proyecto funcione mejor. ¿Son los estándares necesarios para el éxito? Claro que no. Pero ayudan, y ¡necesitamos toda la ayuda que podamos conseguir! Seamos honestos, la mayoría de argumentos contra un estándar en particular vienen de nuestro amor propio, y son una cuestión de gusto.

1.1.4. Haciendo cumplir el estándar

En caso de que haya problemas serios con el estándar, éstos deben comunicarse al grupo de trabajo y a la dirección técnica de Binovo.

1.2. Aceptando una idea

1. ¡Eso es imposible!
2. Quizá sea posible, pero no es importante ni interesante.
3. Es cierto y te lo dije.
4. Ya lo había pensado antes que tu.
5. ¿Cómo podría ser de otra forma?.

Si te acercas a asuntos con prejuicios negativos, trata de mantener una mentalidad abierta. Puede que todavía decidas que el asunto no se sostiene, pero hay un camino que todos debemos recorrer para aceptar algo diferente. Date un tiempo para recorrer el camino.

2. Nombres

2.1. Buscar nombres adecuados

El centro de gravedad de la programación son los nombres. En el pasado la gente pensaba que saber el nombre de alguien les daba poder mágico sobre esa persona. Si puedes adivinar el nombre verdadero para algo, estás dandote a ti y a todos los que vengan detrás de ti Poder Sobre El Código. ¡Y no te rías!

Un nombre debe ser el resultado de un pensamiento largo y profundo sobre la ecología donde vive. Solo un programador que comprende el sistema al completo puede crear un nombre que esté en armonía con el sistema. Si el nombre es adecuado, todo encaja de forma natural; las relaciones son claras, el significado se puede deducir, y el razonamiento en base a expectativas habituales funciona como se espera.

2.1.1. Clases y módulos

- Se debe nombrar una clase o módulo por lo que es. Si no puedes pensar qué es una clase, es un indicio de que no has pensado el diseño lo suficiente.
- Nombres compuestos de más de 3 palabras, son un indicio de que tu diseño está confundiendo varias entidades en el sistema. Revisa el diseño, y las responsabilidades de los objetos.
- Evita la tentación de usar el nombre de la clase base como parte del nombre de la clase derivada. Una clase debe ser autosuficiente. Da igual los ancestros que tenga.

2.1.2. Métodos y funciones

- Normalmente todos los métodos y funciones realizan una acción, por lo que el nombre debe indicarlo empezando por un **verbo infinitivo**. Las funciones que forman parte de un módulo, deberán tener el prefijo del módulo por delante, en mayúsculas y separando las palabras por un subrayado '_'. Esto ayuda a que las funciones y los objetos de datos se distingan fácilmente.

- Perfijos estándares:

    - *Es* / *Esta* – para preguntar sobre algo. Para métodos y funciones que devuelven un booleano.

    - *Get* – obtener un valor, resulta más breve que usar Obtener.

    - *Set* – establecer un valor, resulta más breve que usar Establecer.

  Por ejemplo: esCliente().

2.2. No usar abreviaturas/acrónimos en mayúsculas

- En los casos en los que tengamos un acrónimo en mayúsculas, debemos usar una primera letra mayúscula seguido de minúsculas. No hay ninguna excusa para no cumplir esto.

- Debes usar: getHtmlStatistic.
- No debes usar: getHTMLStatistic, getHTMLstatistic, etc.

Justificación

- Las personas parece que tenemos una intuición muy diferente al hacer nombres que contienen acrónimos. Esta norma de estilo permite que no exista lugar a dudas sobre el nombre correcto.

2.3. Nombres de clases

- (binovo) No usar el caracter subrayado ('_') , (En el PSR-1 punto 3 se indica que se usará "_" para pseudo-namespacing en PHP v5.2.x y anteriores; está guía solo permite el uso de "_" solo si estamos trabajando en un proyecto con un PHP viejo)

2.4. Nombres de métodos y funciones

- Solo pueden contener caracteres alfanuméricos.
- (binovo) Los subrayados ('_') no están permitidos.
- (binovo) Los números están permitidos pero se recomienda evitarlos.
- (binovo) Las funciones pertenecientes a un módulo tendrán como prefijo el nombre del módulo, en mayúsculas y separando las palabras con '_'.
- (binovo) Los métodos y funciones private y protected tendrán el prefijo '_'. La razón es que si un método o función no tiene la visibilidad necesaria, el error se detecta en tiempo de ejecución; sin embargo si les ponemos el prefijo '_' resulta evidente al escribir o leer el código, ya que solo podrán aplicarse sobre $this o parent:: . (Conflicto con PSR-12 4.4, párrafo 3 que lo desaconseja)
- (binovo) En el caso de un módulo
  - PHP 5.3.x+: el módulo tendrá namespace propio. El carácter '_' se prefija a la variable o función privada.

```
namespace \Binovo\Modulo;

function _hacerAlgoPrivado()
{
...
}
```

  - PHP 5.2.x y anteriores: el carácter '_' se prefijará al prefijo del módulo: _MODULO_hacerAlgoPrivado(). Es necesario poner el prefijo de módulo a las funciones privadas para evitar colisiones de nombres entre funciones.

2.5. Variables y parámetros de métodos y funciones

- Las variables miembro private y protected tendrán el prefijo '_'. (Conflicto con PSR-12 4.4; mismo caso que con los nombres de clase)

2.6. Defines / Constantes

- [PSR-1 4-1, binovo para constantes globales] Deben escribirse completamente en mayúsculas, separando las palabras con '_'.

- (binovo) Se debe tratar de definirlos dentro de una clase como variable estática constante (const), evitando define() globales. En caso de hacer defines globales, el nombre de la variable debe tener un prefijo unico que identifique el namespace y el módulo/clase al que corresponde.

2.7. Elementos de array (binovo)

(no tengo muy claro que esto sea buena idea de todas formas?)

Se escribirán en minúsculas, separando las palabras mediante el caracter de subrayado '_'. Debe evitarse usar el caracter '-', puesto que impide usar el índice sin acotarlo (aunque tampoco permitimos usar índices textuales sin acotar). Ejemplos:

```
$myarr['foo_bar'] = 'Hello';


print "{$myarr['foo_bar']} world";
```

2.8. Nomenclatura de ficheros

Todos los ficheros se nombrarán utilizando mayúsculas para separar los nombres. Por ejemplo: InformeOperarios.tpl, UsuariosVista.php

- (binovo) Proyectos que no usen class autoloaders
    - (binovo) Se establece una excepción respecto a [PSR-1 2.3], indicando que el uso de require_once no supone un side effect, siempre que el fichero requerido no ejecute lógica con side effects. Esto se hace porque sino no es posible cargar clases y módulos sin un class autoloader.
    - (binovo) Los ficheros que sean punto de entrada, tendrán el nombre que corresponda al módulo de la aplicación (Albaran.php). Como excepción, el punto de entrada principal de la aplicación podrá ser index.php, que redirigirá al controlador adecuado (Login.php, etc.)
    - (binovo) Sólo proyectos viejos tipo Idi
        - Los ficheros que contengan una clase, tendrán el mismo nombre que la clase, respetando la capitalización de sus caracteres, seguido del sufijo ".class.php". (AlbaranEntrada.class.php)

- Los ficheros que contengan un módulo, tendrán el mismo nombre que el módulo, seguido del sufijo ".inc.php" (Usuarios.inc.php).

3. Formato del código fuente

3.1. true / false / null

Los valores **true**, **false** y **null** se escribirán siempre en minúsculas, a pesar de que son constantes predefinidas en PHP. Esta norma es una excepción puesto que el resto de constantes (predefinidas o no) siempre se escribirán en mayúsculas.

3.2. Indentación

- (binovo) Se debe indentar tanto como sea necesario, pero no de forma innecesaria. Generalmente debería bastar con 4 o 5 niveles de indentación; sino debería considerarse factorizar el código.

- (binovo) Además de la indentación regular por bloques de las líneas, en ocasiones en interesante alinear zonas interiones de varias líneas contiguas. Por ejemplo:

```
$tabla['cant']       = $tabla0['cant']       + $tabla1['cant'];
$tabla['cant_total'] = $tabla0['cant_total'] + $tabla1['cant_total'];
```

-

3.3. Finalización de archivo PHP

- Los ficheros que contengan código PHP comenzarán con <?php, pero no terminarán con ?>, para evitar introducir espacios en el HTML generado.

3.4. Cadenas de texto (Zend)

- Las cadenas de texto que no contengan variables se deben delimitar mediante las comillas simples: $string = 'Esto es una cadena de texto';

- Cuando una cadena debe contener comillas simples (por ejemplo en una sentencia SQL), entonces es preferible usar comillas dobles a escapar las comillas simples: $query = "SELECT * FROM albaran WHERE albaran_descripcion = 'Prueba'";

- En el caso concreto de cadenas muy largas, como por ejemplo sentencias SQL complejas, está permitido el uso del formato heredoc:

```
$tablaCuenta = <<<EOT
CREATE TABLE IF NOT EXISTS `cuenta` (
`cuenta_id` int(10) NOT NULL auto_increment,
`cuenta_padre_id` int(10) NOT NULL,
`cuenta_nombre` varchar(160) NOT NULL,
`cuenta_fecha_creacion` datetime NOT NULL,
PRIMARY KEY (`cuenta_id`),
KEY `idx_cuenta_padre_id` (`cuenta_padre_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=2
EOT;
```

- La sustitución de variables está permitida de las siguientes dos formas:

- $saludo = “Hola $nombre, bienvenido”;
- $saludo = “Hola {$nombre}, bienvenido”;

- La tercera forma **no** está permitida:
  - $saludo = “Hola ${nombre}, bienvenido”;

3.4.1. Índices de arrays

- Siempre que se use una constante de cadena de texto, se usará la notación de comillas simples.
- Cuando se usen arrays dentro de cadenas, las referencias se acotarán mediante llaves.

Ejemplos

```
$myarr['foo_bar'] = 'Hello';
$elementName = 'foo_bar';
print "{$myarr[$elementName]} world"; // will output: Hello world
print "{$myarr["$elementName"]} world"; // will output: Hello world
print "{$myarr['foo_bar']} world"; // will output: Hello world
```

Ejemplos - incorrectos

```
print "$myarr[$elementName] world"; // will output: Hello world
print "{$myarr['$elementName']} world"; // Undefined index: $elementName
print "$myarr[foo_bar] world"; // will output: Hello world
print "{$myarr[foo_bar]} world"; // notice: foo_bar constant not defined
print "$myarr['$elementName'] world"; // parse error
print "$myarr["$elementName"] world"; // parse error
```

3.5. Política de paréntesis de palabras clave de control y funciones

- (binovo) No se deben usar paréntesis en una sentencia return a no ser que sea necesario, en cuyo caso irá con un espacio de distancia a la palabra clave return.

3.6. Clases

- (binovo) La declaración de las variables debe realizarse al principio de la clase, antes de la declaración de los métodos.
- (binovo) Las variables deberán estar en orden alfabético ascendente, ignorando el “_” en el caso de que lo tengan.
- (binovo) Los métodos se declararán en el siguiente orden:

1. Constructor, si existe. Se usará el nombre __construct() de PHP5, y no el nombre de la clase (estilo PHP4).

2. Destructor, si existe: __destruct()
3. Métodos miembro, en orden alfabético ascendente.

```
/**
 * Documentation Block Here
 */
class SampleClass
{
    public    $age;
    protected $_lastname;
    private   $_name;

    private function _getAge()
    {
        return $age;
    }

    public function setName()
    {
        return $_name;
    }
}
```

3.7. Definición de métodos y funciones

(binovo) En caso de que los parámetros sean objetos o arrays, es obligatorio indicarlo en el prototipo. (en caso de estar usando PHP 7, se hará lo propio para los valores escalares)

```
Class Foo
{
      public function FooBar(array $lista, Clase $objeto, $numero)
      {
      }
}
```

3.8. Formato *If Else*. Formato de condiciones

En caso de que la comparación se realice con una constante, se debe poner siempre la constante en el lado izquierdo de la condición:

```
if (6 == $errorNum)
```

Si olvidas uno de los caracteres =, el parse detectará el error por ti. También es conveniente que el valor con el que se está comparando esté al comienzo de la condición

y no al final de una larga condición. Cuesta un poco adaptarse a este formato, pero resulta muy útil.

3.9. Formato *switch*

- El caso *default* siempre debe estar presente, y mostrar un error si no se debiera llegar, en caso de que se llegue al mismo.

Ejemplo

```
switch ($dato) {
    case 1:
        ...
        // Continua en siguiente case
     default:
}
```

3.10. Uso de *continue, break*

Continue y break son en realidad gotos disfrazados, por los que se procurará evitarlos, teniendo en cuenta que en ciertos casos son la construcción más adecuada:

```
for ($nombres => nombre) {
    if (!esAmigo($nombre)) {
        continue;
    }
    saludarAmigo($nombre);
}
$tengoUnSuperAmigo = false;
for ($nombres => nombre) {
    if (esMiUnicoMejorAmigoDelAlma($nombre)) {
        $tengoUnSuperMaigo = true;
        break;
    }
}
```

En especial, debe ponerse mucho cuidado en no mezclar break y continues en el mismo bucle.

3.11. ?: (binovo)

El problema con esta estructura de control es que tendemos a querer poner demasiadas cosas entre la ? y la : . Hay que tratar de evitarlo. Si necesitamos más de una línea, definitivamente debemos reemplazarlo por un if/else.

3.12. Evitar asignaciones en condiciones (binovo)

Realizar asignaciones en condiciones suele generar problemas. Evítalo. Si encuentras un caso en el que parece que hacer esa asignación en una condición simplifica el código, consulta primero con un compañero para ver si no hay otra forma mejor de hacerlo.

```
if ($a = obtenerNumero()) {
	echo 'No es cero';
} else {
	echo 'Es cero';
}
```

Esta regla aplica a los operadores ++ y --. No los mezcles con llamas a función, condiciones, etc.

```
$i = 0;
while ($i < 10) {
	hacerAlgo($i++);
}
```

3.13. No depender del test diferente de 0 (binovo)

En las condiciones siempre debemos comparar con algo, a no ser que estemos comprobado una función que devuelve un valor de **tipo** boolean.

3.14. Nada de números mágicos (binovo)

Un número mágico es una constante desnuda en el código fuente. Es mágico porque nadie tiene ni idea de qué significa, incluyendo el autor que lo escribió hace menos de 3 meses. Por ejemplo:

```
if (22 == $foo) {
	comezar_guerra_termonuclear();
} elseif (19 == $foo) {
	devolver_toneladas_de_dinero();
} elseif (16 == $foo) {
	ciclar_hasta_el_infinito_y_mas_alla();
} else {
	llorar_porque_me_he_perdido();
}
```

¿Qué quieren decir el 22 y el 19? Si hubiese un cambio de numeros o simplemente estuvieran mal, ¿cómo podríamos saberlo?

Debemos reemplazar ese tipo de números con constantes con un nombre que signifique algo.

3.15. Formato de array (binovo)

Después de la coma de separación entre cada elemento del array, se debe dejar un espacio (exceptuando alineaciones). Ejemplos:

```
$a = array('manzanas', 'peras', 'limones');
```

3.16. Asignaciones (binovo)

Debe haber un espacio entre la variable y el operador de asignación y entre el operador de asignación y la expresión que se asigna.

```
$a = $b + $c;
$ok = hacerAlgo();
```

4. Estilo funcional

4.1. Política de comprobación de valores de retorno

- Debe comprobarse el valor de retorno de todas las llamadas a métodos y funciones que devuelvan un valor.
- Se debe incluir el mensaje de error para todo error del sistema.

4.2. No realizar trabajo real en el constructor

En un constructor no se debe realizar ningún *trabajo* real, puesto que no pueden devolver errores.. Solo se deben inicializar las variables y/o realizar tareas que no puedan fallar.

Si es necesario, se puede crear un método Abrir() para completar la construcción. Se llamará a Abrir() después de instanciar el objeto.

Otra alternativa sería usar algún patrón tipo Builder o Factory.

4.3. Hacer las funciones reentrantes

Las funciones no deben tener variables estáticas que eviten que sean reentrantes.

4.4. Parámetros de entrada GET/POST

Los parámetros de entrada se deberán acceder siempre mediante $_GET y $_POST, y los de sesión mediante $_SESSION. **Nunca** debe accederse a las variables directamente, y en los servidores de desarrollo y producción se deshabilitará esa opción (register_globals = off).

Los parámetros de entrada se leerán **todos a la vez en el inicio del script**, tras comprobar la autenticación, y tras filtrar su contenido (asegurarnos de que los valores numéricos solo contienen números, que no hay caracteres inesperados en cadenas, etc.) se almacenarán en variables locales que usaremos en adelante. El motivo es realizar todo el control de limpieza de los datos que hemos recibido a la vez, para que no se nos olvide y no nos metan datos envenenados al programa.

Se debe evitar el acceso a las variables superglobales desde cualquier punto que no sea un punto de entrada de la aplicación. También se debe evitar pasar una copia de una variables superglobal a una función o método:

```
$datos = $_GET;
hacerCosas($datos);
```

En este caso estamos pasando $_GET a la función hacerCosas, aunque sea de forma encubierta.

4.5. Incluyendo código (PEAR)

Usar siempre require_once (inclusión incondicional) o include_once (inclusión condicional) para incluir otros ficheros. De esta forma evitamos que el mismo fichero se cargue más de una vez, ralentizando la ejecución.

4.6. Variables estáticas y globales

Se deben evitar las variables estáticas y globales. En caso de que sea imprescindible tener una variable global, debe considerarse el uso de una clase estática en vez de dejar la variable en el ámbito global.

4.7. Acceso a bases de datos

El acceso a bases de datos se hará por norma usando PDO y sentencias preparadas. El motivo de usar PDO con sentencias preparadas es evitar en la medida de lo posible problemas de seguridad de injección de código SQL. El siguiente código es potencialmente inseguro:

```
$sql = “SELECT * FROM item WHERE item_nombre LIKE '$cadena';
$db->execute($sql);
```

Un usuario malicioso podría hacer que $cadena sea:

'; DELETE * FROM tabla WHERE 1<>0 OR '' = '

eliminando efectivamente todos los registros de tabla.

En lugar el código anterior deberíamos hacer:

```
$sql = “SELECT * FROM item WHERE item_nombre LIKE :cadena”;
$param = array(':cadena' => $cadena);
$stmt = $db->prepare($sql);
if (!is_object($stmt)) {
      // tratar error, en este caso devolvemos false
      return false;
}
$ok = $stmt->execute($param);
if (false === $ok) {
      return false;
}
```

4.8. Logs

Para hacer logs de la aplicación usaremos syslog, evitando hacer echos directos a la salida del script. En caso de encontrar problemas hay que logear la máxima información posible para su posterior diagnóstico, y al usuario mostrárle una pantalla de error que pueda comprender.

5. Documentación

5.1. Comentarios (PEAR)

La documentación del código debe seguir la convención PHPDoc, similar al Javadoc http://phpdoc.org/

Los comentarios no deben ser un reemplazo a un código limpio, claro y con buenos nombres de variables y funciones.

Se permiten comentarios estilo C (/* */) y C++ comments (//), pero no los de estilo Perl/shell (#).

5.2. Comentarios sobre los comentarios

5.2.1. Documentar las decisiones

Los comentarios deben documentar las decisiones tomadas. En cada punto donde se tuvo que tomar una decisión, se debería poner un comentario indicando la decisión tomada y los motivos.

5.2.2. Hacer explícito lo que no es obvio

Se deben comentar explícitamente las variables que cambien fuera del flujo normal de control, u otro código que podría romperse en matenimiento futuro. Se usan palabras clave para marcar problemas o potenciales problemas.

5.2.2.1. Palabras clave

- **:TODO:** Hay algo sin terminar, no lo olvides
- **:BUG: [bugid] topic**
  Problema conocido en este punto
- **:WARNING:**
  Cuidado con algo.

5.2.3. Prohibido el código comentado (que no los comentarios al código)

El código terminado, no tiene código fuente comentado, **nunca**. El código fuente comentado hace más largo y difícil leer el código, y descomentarlo puede introducir bugs debido a que mientras ha estado comentado el contexto ha cambiado.

```
/* $a = $b * $c + 5 */ <--- Código comentado
$r = $a + $c
```

Por lo tanto, y debido a que el código comentado es una fuente de problemas, y no resulta de ayuda, nunca deberemos tenerlo en el código de los repositorios.

5.3. Archivos (Zend)

Cada fichero que contenga código PHP debe tener un docblock al principio con los siguientes contenidos mínimos. Se pone un ejemplo para la aplicación IDI:

```
/**
 * Descripción breve para el archivo
 *
 * Descripción larga para el fichero (si es necesario)
 *
 * @copyright 2008,2009 Binovo IT Human Project, S.L.
 * @license http://www.opensource.org/licenses/bsd-license.php New-BSD
 */
```

5.4. Clases (Zend)

Cada clase debe tener un docblock antes de la definición de la clase con los siguientes contenidos mínimos:

```
/**
 * Descripción breve para la clase
 *
 * Descripción larga para la clase (si es necesario)
 */
```

Esto no excluye el que se tenga que incluir también el docblock de fichero al principio del fichero. Puede resultar que el docblock de fichero y clase estén seguidos, o bien que estén separados por declaraciones como defines y includes/requires.

5.5. Funciones y métodos

Cada función y método, debe tener un docblock que contenga como mínimo lo siguiente:

- Una descripción de la función
- Todos los parámetros
- Todos los valores de retorno posibles

No es necesario usar el tag "@access", puesto que es una información ya conocida de los modificadores "public", "private", o "protected".

Si una función o método puede lanzar una excepción, se debe usar @throws para todas las clases de excepción conocidas:

```
@throws clasedeexcepcion [descripcion]
```

5.6. Documentación de interfaz e implementación

Hay dos destinatarios principales de la documentación:

- Usuarios de la clase/módulo
- Programadores de la clase/módulo

Pensando con un poco de antelación, podemos extraer los dos tipos de documentación directamente del código fuente.

5.6.1. Usuarios de la clase/módulo

La información se extraerá directamente de los encabezados de fichero, clases, metodos y funciones. Cuando se complete esa documentación, se incluirá solo información necesaria para los usuarios de la clase o módulo. No explicar detalles de algoritmos a no se que sean necesarios para usar la clase/módulo.

5.6.2. Programadores de la clase/módulo

La información sobre el detalle de la implementación se recogerá en comentarios de la implementación, que deben recoger información sobre el algoritmo y las decisiones de diseño.

6. Referencias

PHP-FIG

- http://www.php-fig.org/

Basado originalmente en

- http://www.dagbladet.no/development/phpcodingstandard/, versión del 2003-02-17
- PEAR Coding Style, http://pear.php.net/manual/en/standards.php
- Zend Framework Coding Style, http://framework.zend.com/manual/en/coding-standard.html

B

Proiektuaren jarraipen dokumentua

Data	Ikasketa		Kudeaketa			Diseinua	Garapena			Dokumentazioa		Denera
	REST API	API Plat.	Jarraipena	Plangintza	Bilera		API	Test auto.	Komando	Memoria	Defentsa	
21/03/18					5							5
21/03/22	4	4										8
21/03/23		3				2	3					8
21/03/24-26		3	0,25				5					8,25
21/03/29			0,25	3						1		4,25
21/03/30			0,25		2							2,25
21/03/31			0,,25							3		3
21/04/06			0,25	2						1,5		3,75
21/04/07			0,25	3						1,5		4,75
21/04/08			0,25							2,5		2,75
21/04/09			0,25							3		5
21/04/10			0,25							1,5		1,75
21/04/11			0,25							1		1,25
21/04/12		2	0,75			1	3,5			0,5		7,75
21/04/13			0,5		0,75		6,75					8
21/04/14			0,25			0,5	4,75			2		7,5
21/04/15			1,5			0,75	4,5					6,75
21/04/16			0,25			2	2			1		5,25
21/04/17												
21/04/18												
21/04/19			0,5			1	5,75					7,25
21/04/20		1	0,25			1	4,5					6,75
21/04/21			0,25			0,75	5,25					6,25
21/04/22			0,25			1	5,25					6,5
21/04/23		1,5	0,25				2,5					4,25
21/04/24												
21/04/25												
21/04/26		2,25	0,5				2,5			1		6,25
21/04/27		1	0,25			2				1,5		4,75
21/04/28			0,75		2,5		4,75					8
21/04/29		0,5	0,5				3,75	2,5		1		8,25
21/04/30		1	0,25				1	2				4,25
21/05/01												
21/05/02												
21/05/03			0,5					5				5,5
21/05/04			0,25					6				6,25
21/05/05			0,25					6,5				6,75
21/05/06			0,25				1,25	3,5				5
21/05/07			0,25					3,5				3,75
21/05/08												
21/05/09												
21/05/10			0,5		0,5		2,75	5,3				9,05
21/05/11			0,25					7,5				7,75
21/05/12			0,25					7,75				8
21/05/13		0,5	0,25					3,25	3,25			7,25
21/05/14			0,25				1,5	3,25	1			6
21/05/15												
21/05/16												
21/05/17			0,5					3	5			8,5
21/05/18			0,5					1,5	5,75			7,75
21/05/19			0,25					0,5	7,5			8,25
21/05/20			0,25					0,5	6,5			7,25
21/05/21			0,25						5			5,25
21/05/22												
21/05/23												
21/05/24			0,5							4,25		4,75
21/05/25			0,25							4,25		4,5
21/05/26			0,25							5,25		5,5
21/05/27			0,25							2,5		2,75
21/05/28			0,25							4		4,25
21/05/29												
21/05/30			0,25							4		4,25
21/05/31			0,5							4		4,5
21/06/01			0,25							4,25		4,5
21/06/02			0,25							4,5		4,75
21/06/03			0,25							3,5		3,75
21/06/04			0,25							2,5		2,75
21/06/05												
21/06/06												
21/06/07			0,5							1,25		1,75
21/06/08			0,25							1,75		2
21/06/09					0,5					1		1,5
21/06/10			0,25							1		1,25
21/06/11			0,25							1		1,25
21/06/12												

21/06/13												
21/06/14			0,25							2		2,25
21/06/15												
21/06/16			0,25							1,7		1,95
21/06/17			0,25							1		1,25
21/06/18												0
21/06/19												
21/06/20												
21/06/21												
21/06/22												
21/06/23												
21/06/24												
21/06/25												
21/06/26												
Data	Ikasketa		Kudeaketa			Diseinua	Garapena			Dokumentazioa		Denera
	REST API	API Plat.	Jarraipena	Plangintza	Bilera		API	Test auto.	Komando	Memoria	~~Defentsa~~	
	4	19,75	19,25	8	11,25	12	70,25	61,55	34	75,7	0	317,5
	23,75		38,5			12	165,8			75,7		317,5

Estimazioa	5	15	20	15	15	10	80	50	50	70	0	330
	20		50				180			70		
Desbiderapena	-1	4,75	-0,75	-7	-3,75	2	-9,75	11,55	-16	5,7	0	-12,5
	3,75		-11,5				-14,2			5,7		

Bibliografia

[1] Alice - expressive fixtures generator. https://github.com/nelmio/alice. Ikusi 32 orrialdea.

[2] Boilerplate code. https://en.wikipedia.org/wiki/Boilerplate_code. Ikusi 14 orrialdea.

[3] Box project. https://github.com/box-project/box. Ikusi 15 orrialdea.

[4] Dependency injection. https://en.wikipedia.org/wiki/Dependency_injection. Ikusi 47 orrialdea.

[5] Git documentation. https://git-scm.com/docs. Ikusi 15 orrialdea.

[6] Json documentation. https://www.json.org/json-en.html. Ikusi 14 orrialdea.

[7] Respresentational state transfer. https://en.wikipedia.org/wiki/Representational_state_transfer. Ikusi 19 orrialdea.

[8] Restful api. https://restfulapi.net/. Ikusi 19 orrialdea.

[9] Rsnapshot webpage. https://rsnapshot.org/. Ikusi 17 orrialdea.

[10] Rsync webpage. https://rsync.samba.org/. Ikusi 17 orrialdea.

[11] Test fixtures in software. https://en.wikipedia.org/wiki/Test_fixture#Software. Ikusi 32 orrialdea.

[12] Binovo IT Human Project. Binovo. https://www.binovo.eus/. Ikusi 1 orrialdea.

[13] Binovo IT Human Project. Elkarbackup documentation. https://docs.elkarbackup.org/docs/introduction.html. Ikusi 17 orrialdea.

[14] Euskal Herriko Agintaritza Aldizkaria (BOPV). Ticketbai hitzarmena. https://www.euskadi.eus/y22-bopv/es/bopv2/datos/2019/07/1903350a.pdf. Ikusi 3 orrialdea.

[15] Eclipse Foundation. Eclipse documentation. http://help.eclipse.org/2020-12/index.jsp. Ikusi 15 orrialdea.

[16] Eclipse Foundation. Egit documentaton. https://www.eclipse.org/egit/. Ikusi 15 orrialdea.

[17] Ibai Imaz. Test motak. https://ibaiimaz.github.io/tdd-hastapena-testak/. Ikusi 31 orrialdea.

[18] Martin Fowler. Givenwhenthen pattern. https://martinfowler.com/bliki/GivenWhenThen.html. Ikusi 33 orrialdea.

[19] Martin Fowler. Objectmother. https://martinfowler.com/bliki/ObjectMother.html. Ikusi 51 orrialdea.

[20] PHP Documentation Group. Php documentation. https://www.php.net/manual/en/, 2021. Ikusi 13 orrialdea.

[21] Symfony SAS. *Symfony Documentation.* Symfony SAS. Ikusi 13 orrialdea.