



Latvijas Universitāte

Datorikas fakultāte

# Modulārā viedās mājas sistēma

Kursa darbs

**Autors:**

Iļja Gubins

*(st. apl. nr. ig11075)*

**Darba vadītājs:**

Leo Seļāvo,

Prof. Dr. dat

Latvijas Universitāte

Rīga, 2014

## **Anotācija**

Šī darba mērķis – analizēt un projektēt viegli paplašināmu modulāru viedās mājas sistēmu, kā arī izveidot prototipu. Mans darbs sastāv no šādām daļām:

### **Sistēmu projektēšana:**

- Analizēt un salīdzināt esošo risinājumus
- Izvēlēties nepieciešamo funkcionalitāti
- Augsta līmeņa dizains

### **Prototipa izstrāde:**

- Zema līmeņa dizains
- Tehnoloģiju salīdzinājums un izvēle
- Prototipa implementācija

Viedās mājas sistēma ļaus gala lietotājiem ietaupīt gan laiku, gan naudu, automātiski (izmantojot iepriekš konfigurētus triggerus un limitus) vai manuāli (ar speciāli izveidoto mājas lapu) kontrolēt pieejamas mājas ierīces. Mājas lapā būs arī iespēja apskatīt vēstures datus, uzstādīt un konfigurēt pievienotas ierīces un norādīt triggerus un limitus.

**Atslēgvārdi:** viedās māja, bezvadu sensoru tīkli, sensors, aktuātors, pārraudzība, mājas vadība

## **Abstract**

Main purpose of this work – to analyze and design easily extendable modular smart home system, aswell as create a prototype of it. My work consists of the following parts:

### **System design:**

- Analyze and comparison of the existing solutions
- Choice of required functionality
- High level design

### **Prototype creation:**

- Low level design
- Comparison and choice of the most fitting technologies
- Implementation of the prototype

Smart home system will allow end users to save both time and money, by automatically (via previously configured triggers and thresholds) or manually (via special web-page) controlling connected home devices. Web-page will also allow to view history data, setup and configure connected devices and specify triggers and thresholds.

**Keywords:** smart home, wireless sensor network, sensor, actuator, monitoring, home management

## **Аннотация**

Главная цель этой работы – проанализировать и спроектировать легко расширяемую, модулярную систему умного дома, а так же создать прототип такой системы. Моя работа состоит из следующих частей:

### **Проектирование системы:**

- Анализ и сравнение существующих решений
- Выбор необходимого функционала
- Проектирование высокого уровня

### **Создание прототипа:**

- Проектирование низкого уровня
- Сравнение и выбор подходящих технологий
- Имплементация прототипа

Система умного дома позволит конечным пользователям экономить время и деньги, управляя устройствами дома в автоматическом режиме (посредством настроенных условий) и вручную, через удобную веб-страницу, на которой кроме управления будет так же доступны история показаний, настройки подключенных устройств и управление условиями-триггерами.

**Ключевые слова:** умный дом, беспроводные сенсорные сеть, сенсор, актуатор, мониторинг, управление домом

# Saturs

Modulārā viedās mājas sistēma.....	1
Anotācija.....	2
Abstract.....	3
Аннотация.....	4
Apzīmējumu un jēdzienu apraksts.....	7
Ievads.....	8
1. Sistēmu projektēšana.....	9
1.1. Esošo risinājumu analīze.....	9
1.1.1. X10.....	9
1.1.3. Z-Wave.....	9
1.1.3. Zigbee.....	9
1.1.5. Salīdzinājums.....	10
1.2. Funkcionalitātes analīze.....	11
1.2.1. Sensoru rādījumi.....	11
1.2.2. Aktuatori.....	11
1.2.3. Notikumi.....	11
1.2.4. Vestūre.....	11
1.2.5. Vairāki lietotāji.....	11
1.2.6. Viegla jauno ierīču instalācija.....	11
1.2.7. Sistēma pieejama attālināti.....	11
1.3. Augstākas līmeņa sistēmas dizains.....	12
2. Prototipa izstrāde.....	13
2.1. Zema līmeņa dizains.....	13
2.2. Tehnoloģiju salīdzinājums un izvēle.....	13
2.2.1. Datoraparātūras tehnoloģijas.....	14
2.2.1.1. Bāzes stācija.....	14
2.2.1.2. Mezgli.....	15
2.2.1.3. Radio modulis.....	15
2.2.2. Programmatūras tehnoloģijas.....	16
2.2.2.1. Datubāze.....	16
2.2.2.2. REST API un lietotāja saskarne.....	17
2.2.2.3. Radio modulis.....	17
2.2.2.4. Radio dēmons.....	18
2.3. Prototipa implementācija.....	19

2.3.1. Radio pakešu specifikācija.....	19
2.3.2. Pieejāmi galapunkti.....	19
2.3.3. Sensora mezgls.....	20
2.3.4. Mezglu iestatīšanas protokols.....	21
Secinājumi.....	22
Pielikums 1. Programmatūras koda fragments.....	23
Literatūras avotu saraksts.....	24

## **Apzīmējumu un jēdzienu apraksts**

***Viedās māja*** - dzīvokļa vai mājas automatizācijas virziens, ar kura palīdzību ir iespējams attālināti vadīt siltumapgādes sistēmu, elektrosistēmu vai atsevišķus elementus.

***Mezglu tīkls*** - Tīkls, kurā ir vismaz divi mezgli, kurus savieno divi vai vairāki ceļi.

***Dēmons*** - Fona programma, kas vajadzības gadījumā tiek izmantota datoros, kuri darbojas UNIX vai kādas citas operētājsistēmas vidē. Dēmons veic tādus pakalpojumus kā, piem., elektroniskā pasta ziņojumu maršrutēšanu, un tas parasti darbojas bez lietotāja iejaukšanās.

## Ievads

Šī darba mērķis – analizēt un projektēt viegli paplašināmu modulāru viedās mājas sistēmu, kā arī izveidot prototipu.

Pasaulē jau ir eksistējoši viedās mājas sistēmas risinājumi, un būtu negudri neapskatīt tos. Tāpēc, savā darbā es apskatīju vispopulārākos bezvadu protokolus, kā *Zigbee*, *Z-Wave*, un arī gan vadu, gan bezvadu protokolu *X10*.

Diemžēl, visi apskatīti protokoli ir proprietāri un ir diezgan dārgi. Tāpēc es nolēmu izveidot savu sistēmu. Pēc pirmo apskata, es nolēmu ka tā dzīvotspējīga ideja, jo manuāli izveidot nepieciešamos komponentus ir līdz 10 reizes lētāk nekā pirkt gatavus ierīces. Lai to īstenot dzīvē, bija nepieciešami izveidot specifikācijas un saprojektēt augsta līmeņu funkcionalitāti.

Bija nolemts, ka sistēmai ir jābūt bez vadiem, ar vislētākiem komponentiem, bet pietiekami drošā lai strādāt reālos dzīvokļa apstākļos. Sistēmai jābūt viegli pieejamo lietotāju saskarne, kas atbalstīs arī mobilas ierīces.



# 1. Sistēmu projektēšana

Šajā nodaļā es gribu apskatīt teoriju aiz viedas mājas sistēmas: esošo risinājumi, funkcionalitātes izvēle un manas sistēmas augstākas līmeņa dizains.

## 1.1. Esošo risinājumu analīze

Šajā apakšnodaļā es gribu apskatīt esošo risinājumus un izveidot salīdzinājumu tabulu kur būs apvienota visa informācija par protokoliem.

### 1.1.1. X10

X10 ir viens no pirmajiem protokoliem. X10 protokols bija izveidots 70os gados. Pirmā protokola versija bija tikai caur mājās elektrolīniju un tikai pēc tam, galu galā gāja bezvadu. X10 ir zināms ka protokols ar vismazāko ātrumu un vismazāk iespējam. X10 tehnoloģija jau novecoja un ir ļoti iesakāms instalēt kaut ko saderīgāko ar jaunākiem bezvadu standartiem, jo X10 ar katru gadu ir grūtāk un grūtāk uzstādīt. Bet X10 ir joprojām de facto standarts.

### 1.1.3. Z-Wave

Z-Wave ir bezvadu mājas automatizācijas protokols, kas strādā uz 908.42MHz frekvenču joslā. Tas ir salīdzinoši jauns mājas automatizācijas protokols, bet ir pieaudzis diezgan strauji pēdējos gados. Grupa aiz tā protokola, Z-Wave Alliance, kas tagad apvieno vairāk nekā 1000 dažādām ierīcēm. Viena no galvenajām iezīmēm Z-Wave ir to ka to izmanto mezglu tīklu, kas būtībā nozīmē, ka viens Z-Wave produkts sūt informāciju caur citu, un tā tālāk līdz tas sasniedz paredzēto galamērķi. Tam arī ir ļoti zema enerģijas patēriņš, kas ir ideāli piemērots ierīcēm, kas balstās uz akumulatora enerģiju.

### 1.1.3. Zigbee

ZigBee ir 802 bezvadu sakaru IEEE standarts. Tā patērē ļoti mazu elektroenerģijas daudzumu un izmanto mezglu tīklu, kā Z-Wave. Nedaudz lētāks nekā Z-Wave un vairāk

### 1.1.5. Salīdzinājums

<b>Ipašība</b>	<b>X10</b>	<b>Zigbee</b>	<b>Z-Wave</b>
<b>Komunikācijas metode</b>	Bezvadu, 433 MHz	Bezvadu, 2.4Ghz (IEEE 802.15.4)	Bezvadu, 868.42 MHz
<b>Gads</b>	1975	2005	2008
<b>Protokols</b>	Izmanto vienkāršu bitu struktūru lai pārsūtīt komandas. Nav pareizības pārbaudes.	Izmanto datu paketes līdzīgas Ethernetam. Katrai paketei ir datu pareizības pārbaude.	
<b>Ātrums</b>	1x	250kbit/s	100kbit/s
<b>Instalācija</b>	Papilda aparātūra nav vajagdzīga priekš uzstādīšanai. Sistēmu var uzstādīt gala lietotājs.		
<b>Maks. ierīces vienā tīklā</b>	256	~64000	232
<b>Traucējumu kompensācija</b>	Nav	Ir	Ir
<b>Uzticamība</b>	Pārraidītam komandām nav apstiprinājumu. Ja komanda tiek zaudēta, ne sūtītājs, ne saņēmējs nav noformēts par palaistu garām komandu.	Visam komandām ir saņemšanās apstiprinājumi. Ja tiek konstatēts komandas zaudējums, signāla avots atkārtoti mēģinās atsūtīt komandu. Ja tiek konstatēts ierīces zaudējums vienā signāla maršrutā, komanda var būt novadīta caur citiem ierīcēm.	
<b>Mezglu tīkls</b>	Nav	Ir	Ir
<b>Drošība</b>	Nav	Ir	Ir
<b>Iebūvētā enerģijas taupīšana</b>	Nav	Ir	Ir
<b>Ierīces status aprasīšana</b>	Nav	Ir	Ir
<b>Sākuma komplekta cena</b>	160 €	350 €	400 €

## **1.2. Funkcionalitātes analīze**

Šajā apakšnodaļā es gribu apskatīt iespējamo gudras mājas iespējas un kādi ir vajadzīgi manām projektam.

### **1.2.1. Sensoru rādījumi**

Tiem jābūt vērtības-agnostic, t.i. pieņem jebkurus vērtības, jebkādā formāta – temperatūra, gaisa mitrums, apgaismības, utml. Sensori.

### **1.2.2. Aktuatori**

Jābūt iespēja atsūtīt signālu kas ieslēgs, izslēgs vai citādi modificēs ierīces apakšsistēmu – piemēram ja mezglam būs pievienots releja ar “gudru” lampu, jābūt iespējai ieslēgt, izslēdz un arī mainīt gaismas spožumu.

### **1.2.3. Notikumi**

Sistēmai jāģenerē notikumus kas ir izdoti kad kaut kāds iepriekš definēts nosacījums tiek izpildīts. Piemēram, ja temperatūra tiek pārsniegta 26C, ieslēgts gaismas kondicionieru.

### **1.2.4. Vēsture**

Visiem notikumiem un sensoru rādītājiem jābūt saglabātiem, ar visiem laika zīmogiem.

### **1.2.5. Vairāki lietotāji**

Sistēmai jābūt vairākiem lietotāju līmeņiem – administrators, kas var mainīt, piemēram, sensoru rādījumu intervālu un administrēt sistēmas mezglus; lietotājs, kas var apskatīt vēsturi un arī izmantot mezglus; viesis, kas var apskatīt tikai tiekošo situāciju un tikai nekritiskiem elementiem.

### **1.2.6. Viegla jauno ierīču instalācija**

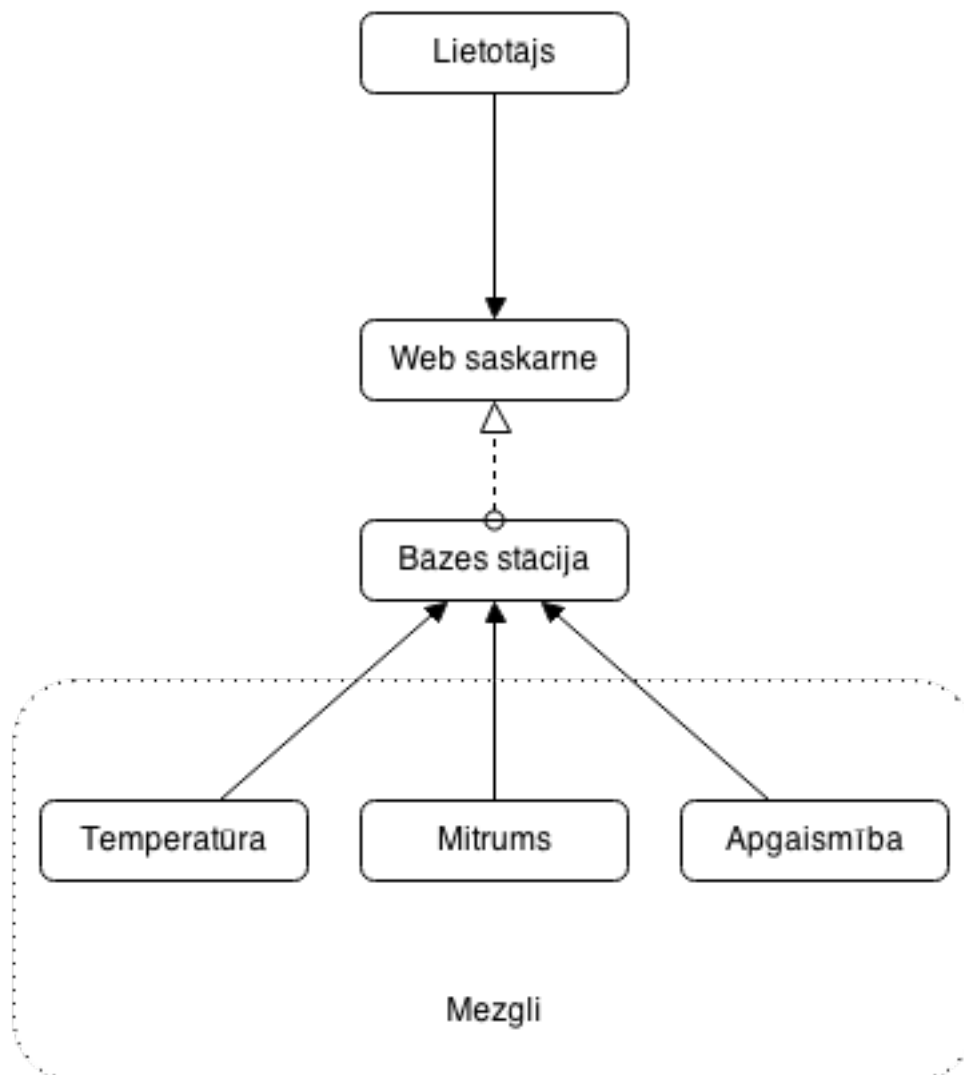
Jauno ierīču pievienošanai jābūt vieglam un ātram, optimāli – ieslēgt ierīce, iekš saskarne apstiprināt ka tas ir tavs mezgls un sakonfigurēt to pēc vajadzībām.

### **1.2.7. Sistēma pieejama attālināti**

Visiem darbībām jābūt pieejamiem arī attālināti, ne tikai kaut kāda vienā vietā vai datorā. Piemēram, internētā, privātā serverī, vai uz bāzes stacijām. Sistēmai arī jābūt ērti

pieejamai no telefonā – vai nu speciālā lietojumprogrammatūrā, vai caur pārlūkprogrammai.

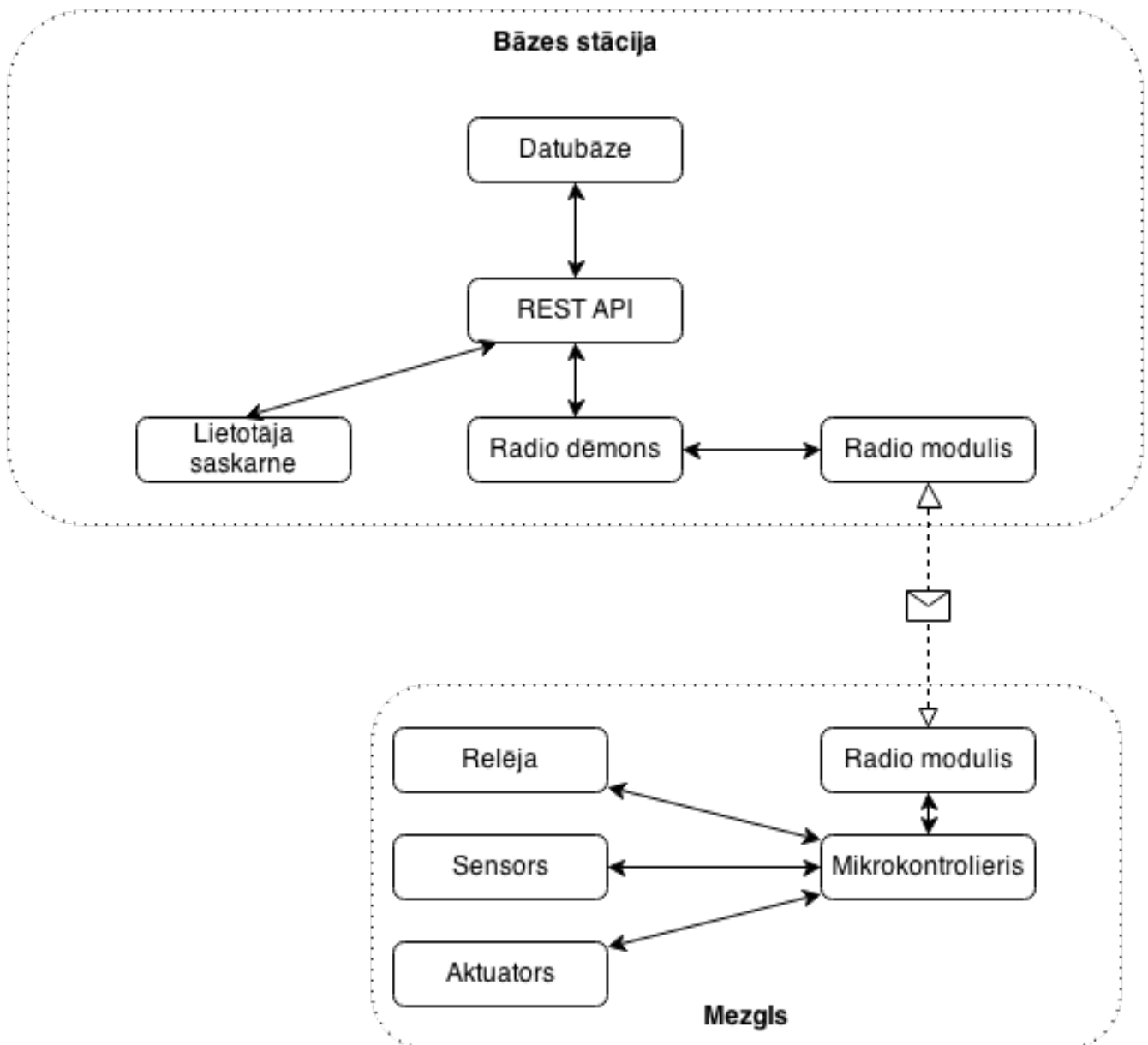
### 1.3. Augstākas līmeņa sistēmas dizains



## 2. Prototipa izstrāde

Šajā nodaļā es gribu apskatīt teoriju aiz viedas mājas sistēmas: esošo risinājumi, funkcionalitātes izvēle un manas sistēmas augstākas līmeņa dizains.

### 2.1. Zema līmeņa dizains



### 2.2. Tehnoloģiju salīdzinājums un izvēle

Šajā apakšnodaļā es gribu apskatīt iespējamo dator aparātūras tehnoloģijas kas varētu būt izmantoti manā implementācijā un argumentācija kāpēc tas tiek izvēlēts.

## 2.2.1. Dator aparatūras tehnoloģijas

### 2.2.1.1. Bāzes stacija

Kā jau bija definēts līdz šim, bāzes stacijai jābūt pietiekami jaudīgai lai atbalstīt interneta pieslēgumu, datubāze, datubāzes API, lietotāja saskarne serveri un jābūt iespējas paplašināt tādu iekārtu ar radio moduli, kaut kāda veidā. Protams, var izmantot tradicionālu personālu datoru, bet tas ir ļoti neefektīvi, gan no enerģijas patēriņa, gan no resursu viedokļa. Visloģiskāk liekas izmantot vienas plates ARM arhitektūras datoru, kur ir pieejami paplašinājuma saskarne, vai nu USB, vai GPIO.

Potenciāli: Raspberry Pi un BeagleBoard Black. Apskatīsim tos vienā tabulā:

Ipašība	Raspberry Pi	BeagleBoard Black	Komentāri
<b>Procesors</b>	Broadcom BCM2835 ARM11 @ 700 MHz	TI Sitara AM3359AZCZ100 Cortex A8 @ 1GHz	Neskatoties uz vienu un to pašu frekvenci, A8 ir apmērām 60% jaudīgāks nekā ARM11
<b>GPU</b>	VideoCore IV	PowerVR SGX530	GPU īsti netiek izmantots, bet tomēr VideoCore IV ir daudz jaudīgāks, nekā PowerVR.
<b>RAM</b>	512 MB SDRAM @ 400 Mhz	512 DDR3L @ 400 Mhz	DDR3L izmanto mazāk elektroenerģijas nekā citi
<b>Datu krātuve</b>	SD kārtas slots	2GB eMMC + microSD slots	eMMC ir tik pats ātrs kā 10. klases SD
<b>Ethernet</b>	10/100M	10/100M	-
<b>USB</b>	2	2	-
<b>Paplašinājumi</b>	12 GPIO, USART, SPI, I2C, CSI, DSI	65 GPIO, SPI, I2C	Iekš BBB ir daudz vairāk GPIO pinus, bet Pi atbalsta vairākus perifērijas formātus
<b>OS</b>	ARMv6 operētājsistēmas, oficiāli – Debian un Arch Linux ARM	Jebkuras ARM operētājsistēmas, oficiāli – Ubuntu un Angstrom	Principā, visērtāk būtu izmantot tīros variantus kā Debian, nevis forkus Ubuntu. Debian nav tik user-friendly, bet vairāk iespējas
<b>Sabiedrības aktivitāte</b>	Ļoti aktīva sabiedrība, vispopulārākais ARM vienas plātes dators	Aktīva sabiedrība, ļoti daudz profesionālus	Šeit Raspberry Pi, neapšaubami, ir līderis

Ipašība	Raspberry Pi	BeagleBoard Black	Komentāri
Cena	\$35	\$200	Gandrīz x6 starpība!
Pieejamība LV	Ir	Ir	-

Neiskaitot to ka priekš Raspberry Pi vēl ir vajadzīga ātrā SD kārte (10. klases), joprojām sanāk ļoti liela starpība cenā. Bija nolēmts, ka pat BBB ir nedaudz labāk atsevišķos gadījumos, Pi vairāk atbilst iespējam un kursa darbas kompetencei.

**Izvēle:** Raspberry Pi

### 2.2.1.2. Mezgli

Mezgliam jābūt pietiekami energoefektīvam, pietiekami jaudīgam lai izmantot radio moduli, ar pietiekami daudz pinus lai pieslēgt sensorus vai aktuātorus. Visērtāk būtu izmantot Arduino vai kaut kādu Arduino klonu. Veiksmīgi, man jau ir Arduino Uno, kas labi atbilst visam prasībām.

**Izvēle:** Arduino Uno

### 2.2.1.3. Radio modulis

Radio modulim jāizmanto brīvo frekvenci, kas ir atļauti visiem; jābūt pievienojam pie izvēlēto ierīcēm – Arduino Uno un Raspberry Pi. Sanāk tā, ka man jau bija pieejami moduli – nRF24L01. Laimīgi, modulim arī ir gatava bibliotēka. Tāpēc,

**Izvēle:** nRF24L01

## 2.2.2. Programmatūras tehnoloģijas

Šajā apakšnodaļā es gribu apskatīt iespējamo programmatūras tehnoloģijas kas varētu būt izmantoti manā implementācijā un argumentācija kāpēc tas tiek izvēlēts.

### 2.2.2.1. Datubāze

Datubāze ir ļoti svarīga daļa, kas ir pamatelements gudras mājas sistēmai. Tā kā ir ļoti daudz datubāzes un gandrīz visi var būt izmantoti, bija nolemts vispirms izveidot prasības:

- Publish – Subscribe modeļa atbalsta, priekš reāllaika notikumiem un izmaiņām
- Raspberry Pi (ARMv6) atbalsta

Pēc apspriedumiem, bija izvēlēti sekojoši datubāzes varianti: Redis, MongoDB, PostgreSQL.

Ipašība	Redis	MongoDB	PostgreSQL
Metamodelis	NoSQL	NoSQL	SQL
ARMv6 Linux atbalsta	Ir	Vajag manuāli kompilēt	Ir
Rezerves kopēšana	Ir	Ir	Ir
Pub/Sub pattern	Ir	Ir	Ir (nedaudz modificēts un zināms kā NOTIFY/LISTEN)
Oficiāli adaptēri	Gandrīz visam valodam		
Īss apraksts	Redis nav datubāze savā pilnā nozīmē, tas drīzāk ir key-value krātuve	MongoDB katrs ieraksts ir JSON objekts. Ir viegli mainīt iekšējo struktūru, bet joprojām var validēt datus, bet ietvara līmenī	Ļoti avansēts, daudz datu tipus un stored procedūras iespējas, bet nav ļoti populārs un ir diezgan lēns, salīdzinot ar citiem SQL un NoSQL datubāzēm

**Izvēle:** MongoDB



## 2.2.2.2. REST API un lietotāja saskarne

REST API būs “līme” starp visām darbībām un datubāze. Kā to izdarīt ir ļoti daudz iespējas, bet vajag arī domāt par resursu patēriņu, jo Raspberry Pi nevar, piemēram, izmantot pilnu Java virtuālo mašīnu, tikai Embedded versiju kā arī uz Linux nevar izmantot .NET. Potenciāli tehnoloģijas: PHP, Go, Node.js.

Go ir jauna programmēšanas valoda kas mani stipri interesē, bet tomēr tā ir pārāk jauna un nestabila. Ļoti daudz būs jāraksta pa jaunam, jo vēl ne visas bibliotēkas ir portēti uz Go.

PHP ir skriptēšanas valoda, kas redzēja ļoti daudz, bet viņai ir mīnusi, kas būs pārāk problemātiski apiet, piemēram, lai izmantot Websockets uz PHP, vajag veidot atsevišķu procesu kas būs atbildīgs par WebSocket savienojumiem. PHP arī nevar strādāt ar savienojumiem pa taisni, vajag vēl vienu procesu, kas būs atbildīga par to, piemēram, Apache vai Nginx.

Node.js, manuprāt, ir ideāla vide. Tā ir pietiekami jauna lai natīvi atbalstīt jaunākas tehnoloģijas (piemēram, Websockets, kas man ir vajadzīgi); tām ir ļoti ērta pakešu sistēma – NPM, kas atvieglo dzīvi un atļauj viegli un ātri instalēt jaunus apakš moduļus; Node.js sabiedrība ir ļoti aktīva un ļoti daudz entuziastus kas veido daudz jaunus moduļus un ietvarus; Node.js izmanto tikai vienu pavedienu savam procesam un izmanto ļoti mazs resursu (kas arī ir ļoti lietderīgs uz Raspberry). Tāpēc bija nolemts izmantot Node.

Bet tas nav viss, priekš Node vajag arī izvēlēties ietvaru. Kā vispiemērotākais ietvars bija izvēlēts SailsJS.

ExpressJS ir diezgan minimāls ietvars. Tas nav slīkti, jo tas atļauj ļoti lokanu programmatūras attīstību, bet tomēr tagad ir vajadzīgs kaut kas cits, kur jau ir vairākas iespējas. Express bija izmests no izvēle ļoti ātri, gandrīz uzreiz pēc izvēlēšanas sakuma, jo viņas konkurentiem ir daudz vairāk iespējas un tie ir vairāk piemēroti priekš REST API.

SailsJS atļauj vienā komandā izveidot resursu, kas būs pieejams uzreiz ar visām pareiziem REST darbībām (scaffolding). SailsJS ir adapteris uz MongoDB, kas natīvi atbalst Pub/Sub. SailsJS arī atļauj izveidot lietotāja saskarne, ne tikai

**Izvēle:** Node.JS un SailsJS ietvars

## 2.2.2.3. Radio modulis

Laimīgi, izvēlētam radio modulim nRF24L01 jau ir izveidota stabila bibliotēka – RF ([github.com/maniacbug/RF24](https://github.com/maniacbug/RF24)). Tam pašam lietotājam arī ir cita bibliotēka – RF24Network

([github.com/maniacbug/RF24Network](https://github.com/maniacbug/RF24Network)), kas pieļauj izveidot mezglu tīklu, bet tagad, pirmā prototipa versijā, es izmantošu vienkāršo variantu – RF24. Alternatīvas priekš RF24 ir, bet diezgan nestabili un nav izturīgi.

**Izvēle:** RF24

#### **2.2.2.4. Radio dēmons**

RF24 bibliotēka ir rakstīta uz C++, bet viņa bija portēta arī uz citām valodām. Tomēr tieši C++ ir vislabāk piemērots darbam ar zemas līmeņa iekārtām kā nRF24L01.

**Izvēle:** C++ ar standartām bibliotēkām + RF24

## 2.3. Prototipa implementācija

Šajā apakšnodaļā es gribu apskatīt gudras mājas implementācijas aspektus.

### 2.3.1. Radio pakešu specifikācija

Tiem jābūt vērtības-agnostic, t.i. pieņem jebkurus vērtības, jebkādā formāta – temperatūra, gaisa mitrums, apgaismības, utml. sensori, kā arī aktuātoru ziņas. Paketēm ir JSON struktūra, jo tā ir vieglāk apstrādāt. Piemērs radio paketēm:

```
{
  fromId: 'mezgluId',
  timestamp: '123456789',
  value: '36.6'
}
```

Kāda tipa mezgls jau ir definēts mezgla instalācijas laikā (un var būt izmainīta caur lietotāja saskarne), tāpēc vajag tikai vērtību, laika zīmogu un sūtītāja identifikatoru.

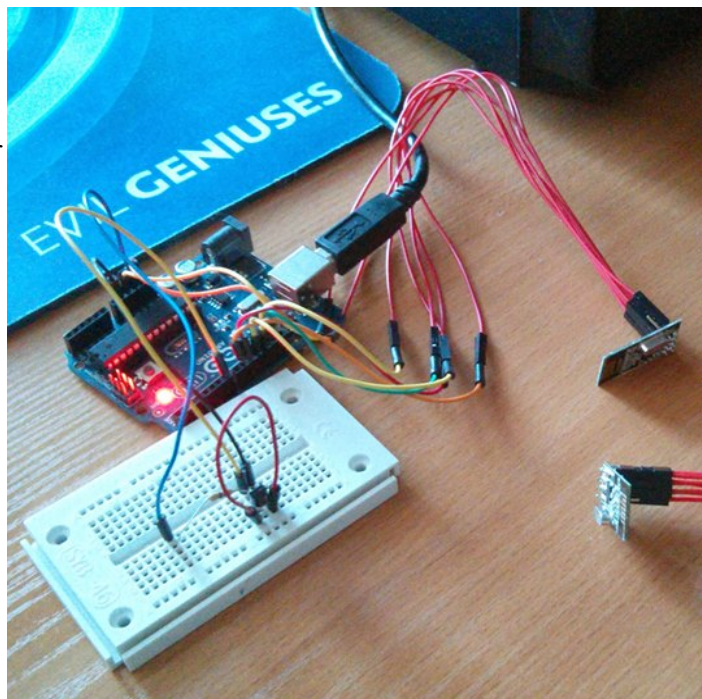
### 2.3.2. Pieejami galapunkti

Galapunkts	Apraksts
<b>GET /devices</b>	Atgriež HTML, kur var apskatīt visus ierīces sistēmā.
<b>GET /devices/id</b>	Atgriež HTML, kur var apskatīt noteiktu ierīci.
<b>POST /devices/found</b>	Jauni, nepievienoti sistēmai ierīces atsuta šeit informāciju par sevi. Atgriež status kodu, veiksmē ja bija pievienots jauna ierīce, neveiksmē ja jauna ierīce nebija pievienots.
<b>POST /devices/pairing</b>	Atgriež status kodu, veiksmē ja slepenais ID kas bija ievadīts ir pareizs, neveiksmē pretējā gadījumā.
<b>GET /devices/setup/id</b>	Atgriež HTML, kur var iestādīt jauno ierīci sistēma. Tai vispirms jābūt pievienotam caur /devices/found.
<b>POST /devices/setup/id</b>	Novirze atpakaļ uz tādu pašu GET.
<b>GET /devices/edit/id</b>	Atgriež HTML, kur var rediģēt ierīces konfigurāciju.
<b>POST /devices/edit/id</b>	Novirze atpakaļ uz tādu pašu GET.
<b>GET /devices/delete/id</b>	Dzēst ierīci no sistēmas. Novirze uz GET /devices.
<b>WS /subscribe/devices</b>	Pieejams tikai ar WebSockets. Abonē websocket notikumus kas saistīti ar ierīcēm.
<b>WS /subscribe/incoming</b>	Pieejams tikai ar WebSockets. Abonē websocket notikumus kas saistīti ar jauniem ienākošiem atskaites no ierīcēm.
<b>POST /report</b>	Galapunkts priekš radio dēmonu. Uz šejieni tas sūt visu informāciju saņemtu no mezgliem, piem. Sensora vērtības. Atgriež status kodu,

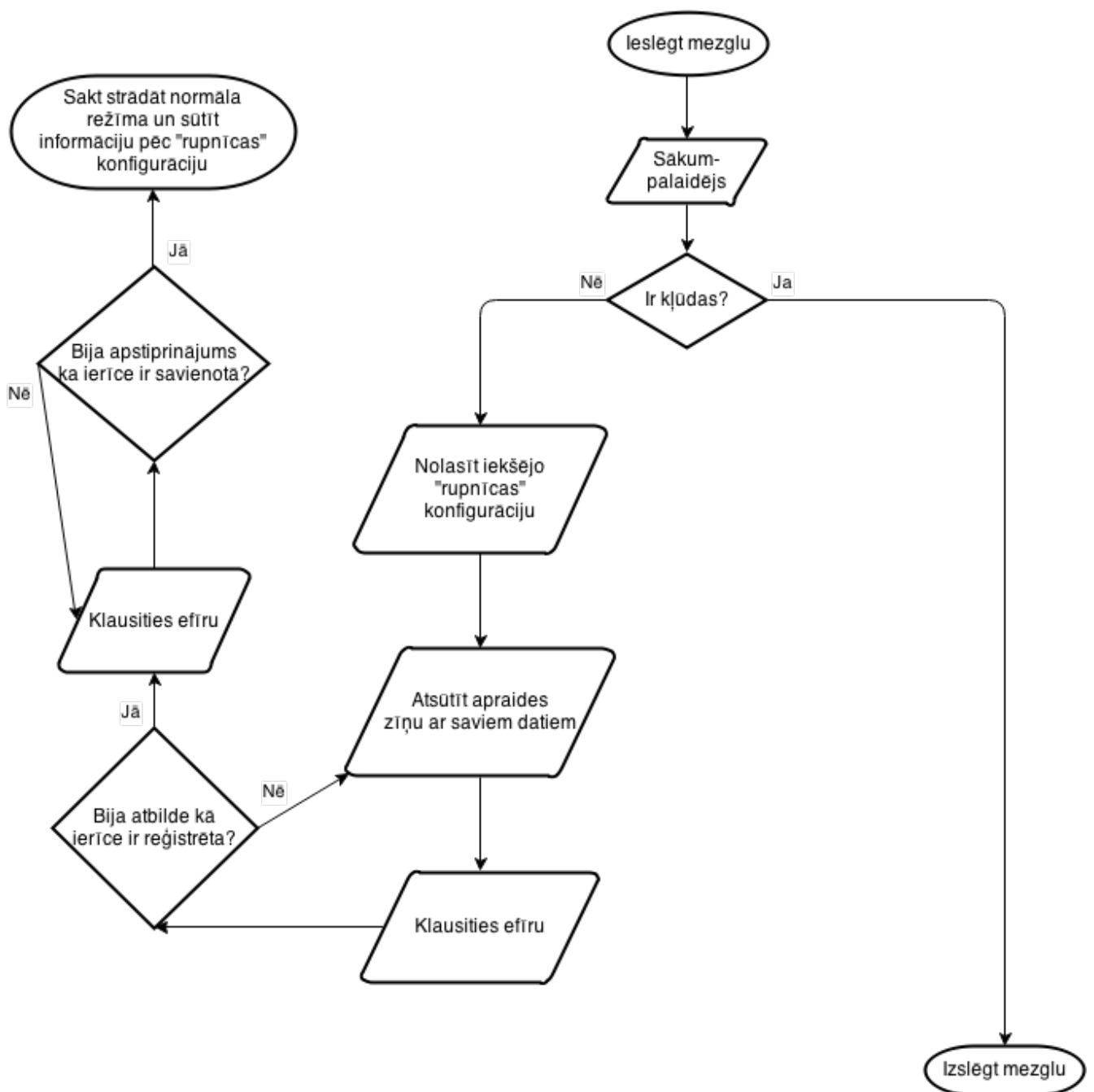
Galapunkts	Apraksts
	veiksme ja vērtība bija pievienota, neveiksme pretējā gadījumā.
<b>GET /history</b>	Atgriež HTML, kur var apskatīt visus pēdējos notikumus sistēmā.
<b>GET /history/id</b>	Atgriež HTML, kur var apskatīt kādu noteiktu notikumu sistēmā.
<b>GET /triggers/</b>	Atgriež HTML, kur var apskatīt visus pievienotus triggerus sistēmā.
<b>GET /triggers/id</b>	Atgriež HTML, kur var apskatīt kādu noteiktu triggeru sistēmā.
<b>POST /triggers/id</b>	Novirze atpakaļ uz tādu pašu GET.
<b>POST /triggers/add</b>	Novirze uz GET /triggers/id
<b>POST /triggers/delete/id</b>	Dzēst triggeru no sistēmas. Novirze uz GET /triggers/.
<b>GET /dashboard</b>	Atgriež HTML, kur var apskatīt tiekošo kopējo statusu sistēmai un arī ātri izmantot kādus populārākus aktuātorus, piem. ieslēgt gaismu.
<b>POST /dashboard/use/</b>	Šeit ir atsūtīta informācija ar to kuru aktuatoru un ko tieši vajag izdarīt. Novirze uz GET /dashboard

### 2.3.3. Sensora mezgls

Fotogrāfijā ir redzams sensoru mezgls – uz maketēs plates ir temperatūras sensors; ar sarkaniem vadiem ir pievienots radio modulis; ar lielo melnu – elektrības vads.



### 2.3.4. Mezglu iestatīšanas protokols



## **Secinājumi**

Savā kursa darba rezultātā, es:

- noprojektēju viedās mājas sistēmu: noanalizēju un salīdzināju esošo risinājumus, izvēlējos nepieciešamo funkcionalitāti, izveidoju augsta līmeņa dizainu;
- izveidoju prototipu tādai sistēmai: sākot no zema līmeņa dizainu, tehnoloģiju salīdzinājumu un izvēlei un izveidoju pašu prototipu.

Atkarība no esošiem risinājumiem, prototips varbūt nav tik profesionāli izskatāms un varbūt nav tik labi notestēts, bet pašizmaksa tādai sistēmai ir ļoti maza, salīdzinot ar gataviem risinājumiem – uz visu sistēmu bija izmantots apmērām 60 eiro. Par vislētāko tādu pašu sistēmu būtu jāmaksā vismaz 100 eiro vairāk.

Prototips labi parāda ka tāda koncepcija strādā un pēc vairāk darbu uz šo, tāda sistēma varbūt izmantota, piemērām manā mājā un saglabāt gan laiku, gan naudu un dod pārliecību ka mājās viss ir labi. Savu darbu rezultātu es gribu turpināt izstrādāt un izmantot savā mājā.

## Pielikums 1. Programmatūras koda fragments

```
/**
 * SubscribeController
 *
 * @description :: Server-side logic for managing pub/sub events.
 */

module.exports = {

  /**
   * `SubscribeController.subscribeDiscoveredDevices()`
   * Allows clients to subscribe for publish events.
   * Used on index page - to monitor new discovered devices.
   */
  subscribeDiscoveredDevices: function(req, res) {
    if (req.isSocket) {
      async.parallel({
        nodes: function(cb) {
          Node.find().sort('added DESC')
            .exec(function(err, nodes) {
              cb(err, nodes);
            });
        },
        discoveredNodes: function(cb) {
          DiscoveredNode.find().sort('added DESC')
            .exec(function(err, discNodes) {
              cb(err, discNodes);
            });
        }
      }, function(err, results) {
        if (err) {
          console.log(err);
          return res.send(500);
        } else {
          Node.watch(req.socket);
          DiscoveredNode.watch(req.socket);

          return res.send(200);
        }
      });
    } else {
      return res.send(403);
    }
  }
};
```

## Literatūras avotu saraksts

1. Node.js dokumentācija
2. SailsJS dokumentācija
3. Z-Wave oficiāla lapa
4. X10 oficiāla lapa
5. ZigBee oficiāla lapa
6. X10, Z-Wave, Zigbee salīdzinājums
7. Maniacbug's RF24 bibliotēka priekš nRF24L01
8. Socket.io dokumentācija



Kursa darbs “Saules bateriju programmēšanas specifika” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti.

Autors: Ilja Gubins

Rekomendēju/nerekomendēju darbu aizstāvēšanai (nevajadzīgo izsvītrot)

Vadītājs: profesors, Dr. dat. Leo Seļāvo

Darbs iesniegts Datorikas fakultātē, 04.06.2014.

Dekāna pilnvarotā persona:

Darbs aizstāvēts kursa darbu komisijas sēdē

Komisija: