Amra jani Javascript ekta functional programming language. Kintu bivinno dhoroner library like mongoose object oriented way te banano hoise. Jokhon amra mongoose library niye kaj korbo tokhon amader application e jotoi functional way te code kori na keno mongoose shoho jekono 3rd party library niye kaj korte gele object oriented coding shomporke dharona rakhte hobe. Class ebong object er bepare dharona rakhte hobe.

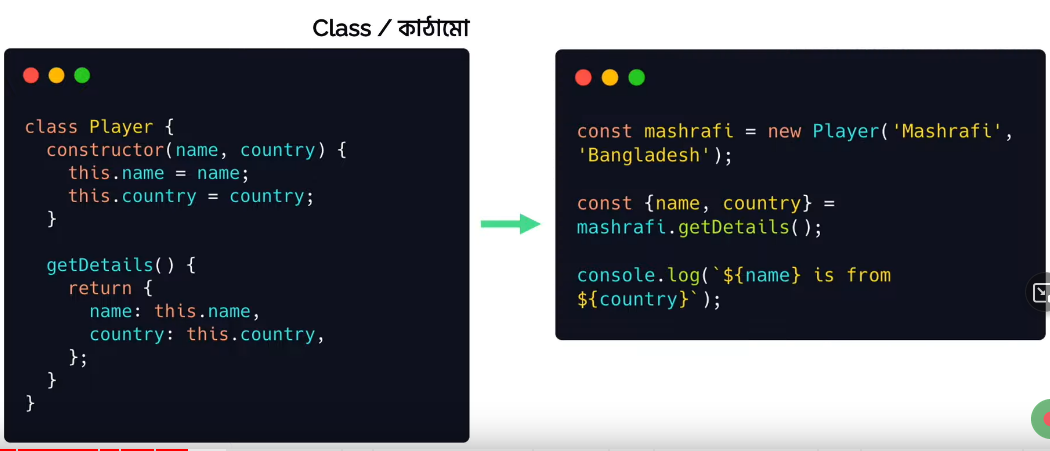
Ajke amra mongoose er instance methods, statics ebong query helpers niye ghataghati korbo. Amra crud operation e post route er moddhe ekta ‘todo’ create korechilam. Tarpor todo.save() call korechilam. Ekhane save() tai hocche ekta instance method.

Instance methods

Instance methods bujhte hole amader object oriented er concept ta ektu jhalai korte hobe. Object oriented concept er mul jinish hocche class.



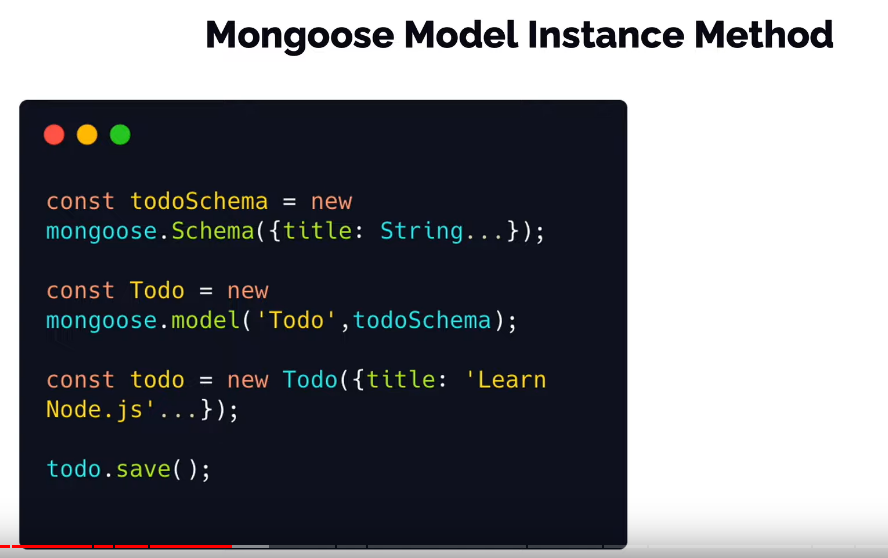
Eta ekta Player class. Player class er moddhe ekta constructor function ache. getDetails() name ekta method ache. Class kono actual jinish na. eta holo ekta kathamo ba blueprint. Ei Player class theke jodi amra ekta actual player banate chai tahole nicher moto kore banate pari.



Ekhane ‘mashrafi’ ekta actual player. Take amra banaisi new Player(‘Mashrafi’, ‘Bangladesh’) call kore. New keyword ta use hoy class er age. Tar mane Player kathamo ke use kore ‘mashrafi’ namok actual player ba object (jinish) create kora hoise. Ekhane ‘mashrafi’ holo Player class er ekta instance. Object oriented programing language er object (which is jinish) ebong vanilla js er object alada.

Erpor mashrafi.getDetails() method ta call korechi. Ei getDetails() hocche instance method. ‘mashrafi’ ekta instance. getDetails() call koraisi ‘mashrafi’ ke diye. Tar mane ‘mashrafi’ instance er ekta method hocche getDetails. Ei karone getDetails ke bola hocche instance method. Instance method er bodyta class er moddhe thakbe. ebong oi class er kono ekta instance (ekhane ‘mashrafi’) oi method ta ke call korbe.

Ekhon amra dekhbo mongoose er khetre instance method ta ki.



Ager din shobar age amra ekta schema banaisilam. Database e amra je data ta rakhbo shei data tar ekta proper structure deyar way hocche schema. Schema ke ekta js object er moto chinta korte pari jetar moddhe amra bole dite pari amar data ta kemon hobe. Data ba document er protita field Kemon hobe. String hobe naki number hobe naki ki hobe --- ei shob er ekta kathamo bole deya jay schema er moddhe. Amra mongoose.Schema({title: …}) call korechilam. Ekhane .Schema() mongoose er ekta function. Etake call kore amra ekta class pacchi. ekarone ‘new mongoose.Schema()’ use korte hoise. Mongoose er Schema kathamo use kore amra ekta todoSchema create korsi. So todoSchema holo amra actual object ta ba actual schema ta.

Erpor new mongoose.model(‘Todo’, todoSchema) kore ekta model banaisilam. Collection er name dilam ‘Todo’. Jetake ultimately ‘todos’ name mongo compass e dekhbo. Mongoose.model() kore ekta model bananor pore je ‘Todo’ ta amra pelam sheta ekta document class. Collection er moddhe je ekekta record thake (object akare postman theke pathai) shegula ke document bole. Orthat ‘Todo’ holo document er ekta kathamo. Erpor new Todo({title: “Lea…}) kore actual document ‘todo’ create korlam. Erpor ‘todo’ instance diye save() call korlam. Ei save() tai holo ekta instance method. Kar instance method? Sheta jante hole dekhte ke etake call korse. ‘todo’ call korse. ‘todo’ ashche ‘Todo’ namok document class ba model theke theke. Amra jani instance methods lekha thake class er moddhe. Orthat save() instance method ta ‘Todo’ class ba model er moddhe ache.

So save() holo mongoose er nijer banay deya built in model instance method.

Ekhon code er maddhome instance method er khela dekhbo. Amra ekta custom instance method banabo. Amra Todo = new mongoose.model(‘Todo’, todoSchema) kore model ta create korechi. Ekhane amra baire theke input disi todoSchema. Orthat ei model er moddhe kono external input ei schemar maddhome dite pari. Mongoose amader ei schema diye jate kore baire theke jabotiyo kaj ei schemar maddhome korte pari. So amader custom instance method gula bananor jonno schema ke use korbo. Ekhon todoSchema.js e jabo. todoSchema ekta actual object ja ashche mongoose.Schema class theke asche. Shei class er ekta property hocche ‘methods’. So ekhon ‘todoSchema.methods’ er moddhe amra ekta object pabo. Shei object er moddhe ekekta property akare custom instance method dibo. Dhori, amader 1st instance method hobe findActive. Etar kaj only active status ala documentgulake pick kora. findActive name ekta property nilam. Etar value dibo ekta function body. Active status ala document pete hole amake mongoose er find method use korte hobe. Ei find amake call korte hoy Todo (which is mongoose.model(‘Todo’)) diye. todoSchema.js e amar .model er access nai. Only mongoose ache ekhane. So findActive er function body theke amra return kore dibo mongoose.model(‘Todo’).find({status : ‘active’}). Ekhane mongoose.model(‘Todo’) hocche Todo. Ekhon findActive ke amra call korbo ki kore? todoHandler.js e notun ekta route nilam. Etar url dilam /getactivetodos. Ei router callback function e firste const todo = new Todo() kore ekta document instance banay nilam. Karon instance method always instance diye call korte hobe. Ekhon ei todo ke diye findActive call korte parbo. Jehetu eta ekta async process shehetu callback function er age async likhte hobe ebong result = await todo.findActive() likhe result ta niye res.json(result) kore dibo.

todoSchema.js …

*const* mongoose = *require*("mongoose");

*const* todoSchema = new mongoose.*Schema*({

  title: {

    type: String,

    required: true,

  },

  description: String,

  status: {

    type: String,

    enum: ["active", "inactive"],

  },

  date: {

    type: Date,

    default: Date.*now*(),

  },

});

todoSchema.methods = {

*findActive*: *function* () {

*return* mongoose.*model*("Todo").*find*({ status: "active" });

  },

};

module.exports = todoSchema;

todoHandler.js …

*// get only active todos*

router.*get*("/getactivetodos", async (*req*, *res*) *=>* {

  try {

*const* todo = new *Todo*();

*const* result = *await* todo.*findActive*();

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

Ekhon instance method er ei example ta callback diye dekhbo. Ekehtre async – await use korbo na.

todoSchema.js …

*const* mongoose = *require*("mongoose");

*const* todoSchema = new mongoose.*Schema*({

  title: {

    type: String,

    required: true,

  },

  description: String,

  status: {

    type: String,

    enum: ["active", "inactive"],

  },

  date: {

    type: Date,

    default: Date.*now*(),

  },

});

todoSchema.methods = {

*findActive*: *function* () {

*return* mongoose.*model*("Todo").*find*({ status: "active" });

  },

*findActiveCB*: *function* (*cb*) {

*return* mongoose.*model*("Todo").*find*({ status: "inactive" }, cb);

  },

};

module.exports = todoSchema;

todoHandler.js ….

*// get only active todos with cb*

router.*get*("/getactivetodoscb", (*req*, *res*) *=>* {

*const* todo = new *Todo*();

  todo.*findActiveCB*((*err*, *data*) *=>* {

    if (err) {

      res.*status*(500).*json*({ error: err });

    } else {

      res.*status*(200).*json*(data);

    }

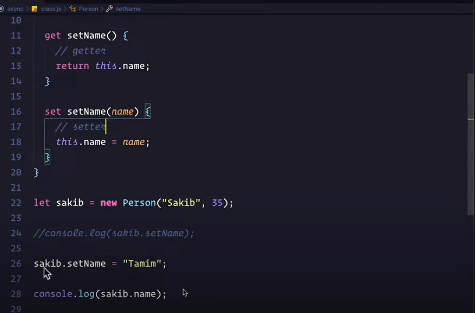
  });

});

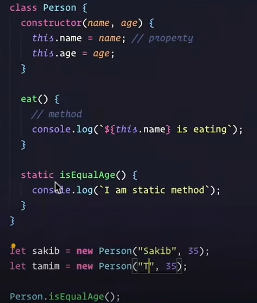
Statics

Ekhon amra dekhbo statics ki. Statics bojhar age amader bujhte hobe kono ekta class er static method ashole ki. Etao object oriented programming er ekta concept. LWS er ‘js prototype inheritance and es6 class’ video te object oriented er js er concept.

Muloto normally amra jokhon class er method define kori tokhon oi method ta ke access korte hole amader first e oi class diye ekta actual object banate hoy. Tarpor oi object diye amra oi method gula ke access kori.



Ekhane sakib.setname use kora hoise. Statics er khetre amader erokom object instantiate korte hobe na. amra static method gulake direct class er name diye (ekhane Person). Ekhane jodi ‘hello’ name ekta static method thakto tahole amra Person.hello() korte partam. ‘sakib’ name ekta object instantiate kore nite hobe na.



Ekhtre method er body defining er shomoy static name ekta keyword diye dite hoy. Ekahne isEqualAge name ekta static method likhechi class er moddhe. Sheta ke ami class er naam diye Person.isEqualAge() call korechi. Etai hocche static method.

Jodi amra instance methods er moto custom static method baire theke dite chai tahole amra kivabe dite pari? Static method er jonno todoHandler.js e ekta notun route nicchi. Etar url hobe /js. todoSchema te ‘title’ name ekta field ache. Ami emon function banate chai jeta ‘js’ title ala todo gulake ene dibe. Static er khetre const todo = new Todo() kora lagbe na. amra direct Todo class diyei static method call korte parbo.

todoSchema er ‘statics’ name ekta property ache. Eta ekta object. Etar ekekta property hishebe ekekta static method dibo. Egulake amra direct Todo model diye access korte parbo. todoSchema.statics er first property dilam findByJs. Etar value dibo ekta function body. Ei body theke return korbo this.find({title : /js/i}). Instance method er khetre mongoose.model(‘Todo’).find() korechilam. Ebar ar sheta korlam na. karon todoHandler.js static method ta call korbe Todo model ta. So Todo diyei jodi call kori tahole todoSchema.statics er property te mongoose.model(‘Todo’) lekhar dorkar nai. Er bodole just ‘this’ likhe dilei hobe. Mongoose.model(‘Todo’) i hocche ekhane ‘this’.

Ami ultimately static method banacchi Todo class er moddhe. Todo class er moddhe ami ‘this’ diye access korte parbo.

TodoSchema.js …

*// static methods*

todoSchema.statics = {

*findByJs*: *function* () {

*return* this.*find*({ title: /js/i });

  },

};

TodoHandler.js …

*// get todos with only js*

router.*get*("/js", async (*req*, *res*) *=>* {

  try {

*const* result = *await* Todo.*findByJs*();

    res.*status*(200).*json*(result);

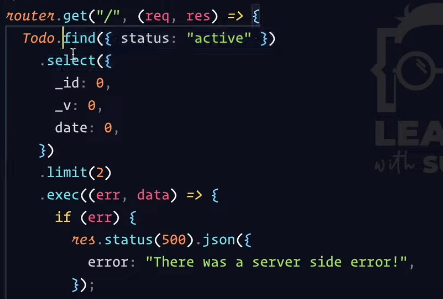
  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }});

Query Helper

Ekhon dekhbo query helper jeta static er motoi simple. Static method er khetre jemon Todo class diye call korechilam temoni query helper o direct model ke diyei call korbo.



Ekhane method chaining kora hoise i.e. Todo.find().select().limit()…. Ei je method gulo ja chaining kora hocche shegula ekekta query helper. .selec(), .limit() egula hocche built in query helper. Amra ekhon custom query helper banabo. Jemon .byLanguage() etar parameter e jodi ‘js’ dei tahole ‘js’ ala shob todo niye ashbe. Jodi ‘react’ dei tahole tahole corresponding todo gula niye ashbe.

todoSchema.js …

*const* mongoose = *require*("mongoose");

*const* todoSchema = new mongoose.*Schema*({

  title: {

    type: String,

    required: true,

  },

  description: String,

  status: {

    type: String,

    enum: ["active", "inactive"],

  },

  date: {

    type: Date,

    default: Date.*now*(),

  },

});

*// instance methods*

todoSchema.methods = {

*findActive*: *function* () {

*return* mongoose.*model*("Todo").*find*({ status: "active" });

  },

*findActiveCB*: *function* (*cb*) {

*return* mongoose.*model*("Todo").*find*({ status: "inactive" }, cb);

  },

};

*// static methods*

todoSchema.statics = {

*findByJs*: *function* () {

*return* this.*find*({ title: /js/i });

  },

};

*// query helpers*

todoSchema.query = {

*byLanguage*: *function* (*language*) {

*return* this.*find*({ title: new *RegExp*(language, "i") });

  },

};

module.exports = todoSchema;

todoHandler.js …

*// get todos with by query helper*

router.*get*("/language", async (*req*, *res*) *=>* {

  try {

*const* result = *await* Todo.*find*().*byLanguage*("css");

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

Package.json….

{

  "scripts": {

    "start": "nodemon index.js",

    "production": "NODE\_ENV=production nodemon index.js"

  },

  "dependencies": {

    "body-parser": "^1.20.0",

    "cookie-parser": "^1.4.6",

    "cors": "^2.8.5",

    "dotenv": "^16.0.3",

    "ejs": "^3.1.8",

    "express": "^4.18.1",

    "mongoose": "^6.7.0",

    "multer": "^1.4.5-lts.1"

  }}

Index.js …

*const* express = *require*("express");

*const* connector = *require*("./data/connection");

*const* router = *require*("./data/TodoHandler");

*const* app = *express*();

app.*use*(express.*json*());

*connector*();

app.*use*("/todo", router);

app.*listen*(3000, () *=>* {

  console.*log*("Listenin on port 3000");

});

.env …

TODO\_URL = "mongodb://localhost:27017/todos"

Connection.js …

*require*("dotenv").*config*();

*const* mongoose = *require*("mongoose");

*const* db\_url = process.env.TODO\_URL;

*const* *connector* = () *=>* {

  mongoose

    .*connect*(db\_url)

    .*then*(() *=>* console.*log*("Connection with databse established"))

    .*catch*((*err*) *=>* {

      console.*log*(err);

    });

};

module.exports = connector;

TodoSchema.js …

*const* mongoose = *require*("mongoose");

*const* todoSchema = new mongoose.*Schema*({

  title: {

    type: String,

    required: true,

  },

  description: String,

  status: {

    type: String,

    enum: ["active", "inactive"],

  },

  date: {

    type: Date,

    default: Date.*now*(),

  },

});

*// instance methods*

todoSchema.methods = {

*findActive*: *function* () {

*return* mongoose.*model*("Todo").*find*({ status: "active" });

  },

*findActiveCB*: *function* (*cb*) {

*return* mongoose.*model*("Todo").*find*({ status: "inactive" }, cb);

  },

};

*// static methods*

todoSchema.statics = {

*findByJs*: *function* () {

*return* this.*find*({ title: /js/i });

  },

};

*// query helpers*

todoSchema.query = {

*byLanguage*: *function* (*language*) {

*return* this.*find*({ title: new *RegExp*(language, "i") });

  },

};

module.exports = todoSchema;

TodoHandler.js …

*const* express = *require*("express");

*const* router = express.*Router*();

*const* todoSchema = *require*("./TodoSchema");

*const* mongoose = *require*("mongoose");

*const* Todo = new mongoose.*model*("Todo", todoSchema);

*// get multiple todos*

router.*get*("/getmultiple", async (*req*, *res*) *=>* {

*//res.send("Get multiple todos");*

  try {

*//const result1 = await Todo.find({});*

*const* result2 = *await* Todo.*find*({ status: "inactive" });

    res.*status*(200).*json*(result2);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*//get a todo by id*

router.*get*("/get/:id", async (*req*, *res*) *=>* {

*//res.send("Get a todo");*

  try {

*const* id = req.params.id;

*const* result = *await* Todo.*findById*(id);

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// get only active todos*

router.*get*("/getactivetodos", async (*req*, *res*) *=>* {

  try {

*const* todo = new *Todo*();

*const* result = *await* todo.*findActive*();

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// get only active todos with cb*

router.*get*("/getactivetodoscb", (*req*, *res*) *=>* {

*const* todo = new *Todo*();

  todo.*findActiveCB*((*err*, *data*) *=>* {

    if (err) {

      res.*status*(500).*json*({ error: err });

    } else {

      res.*status*(200).*json*(data);

    }

  });

});

*// get todos with only js*

router.*get*("/js", async (*req*, *res*) *=>* {

  try {

*const* result = *await* Todo.*findByJs*();

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// get todos with by query helper*

router.*get*("/language", async (*req*, *res*) *=>* {

  try {

*const* result = *await* Todo.*find*().*byLanguage*("css");

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// post a todo*

router.*post*("/createone", async (*req*, *res*) *=>* {

*//res.send("create one todo");*

  try {

*const* todo = new *Todo*({

      title: req.body.title,

      description: req.body.description,

      status: req.body.status,

      date: req.body.date,

    });

*const* result = *await* todo.*save*();

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// post multiple todos*

router.*post*("/createmultiple", async (*req*, *res*) *=>* {

*//res.send("crete multiple todos");*

  try {

*const* result = *await* Todo.*insertMany*(req.body);

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// update a todo by id*

router.*put*("/updatebyid/:id", async (*req*, *res*) *=>* {

  try {

*const* id = req.params.id;

*const* updateData = { $set: { status: "active" } };

*const* options = { new: true };

*const* result = *await* Todo.*findByIdAndUpdate*(id, updateData, options);

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// update multiple todos*

router.*put*("/updatemultiple", async (*req*, *res*) *=>* {

*//res.send("update multiple todos");*

  try {

*const* filterData = { title: "JAVASCRIPT" };

*const* updateData = { status: "active" };

*const* result = *await* Todo.*updateMany*(filterData, updateData);

    res.*status*(200).*json*(result);

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// delete a todo by id*

router.*delete*("/delete/:id", async (*req*, *res*) *=>* {

  try {

*const* id = req.params.id;

*const* result = *await* Todo.*findByIdAndDelete*(id);

    res.*status*(200).*json*({ message: "deleted successfully" });

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

*// delete multiple todos*

router.*delete*("/deletemultiple", async (*req*, *res*) *=>* {

*//res.send("delete multiple todos");*

  try {

*const* filterdata = { status: "active" };

*const* result = *await* Todo.*deleteMany*(filterdata);

    res.*status*(200).*json*({ message: "multiple todos deleted" });

  } catch (error) {

    res.*status*(500).*json*({ error: error });

  }

});

module.exports = router;

inside ‘data’ folder => connection.js , TodoHandler.js, TodoSchema.js