# BAB III Metodologi

## III.1. Metode Penelitian

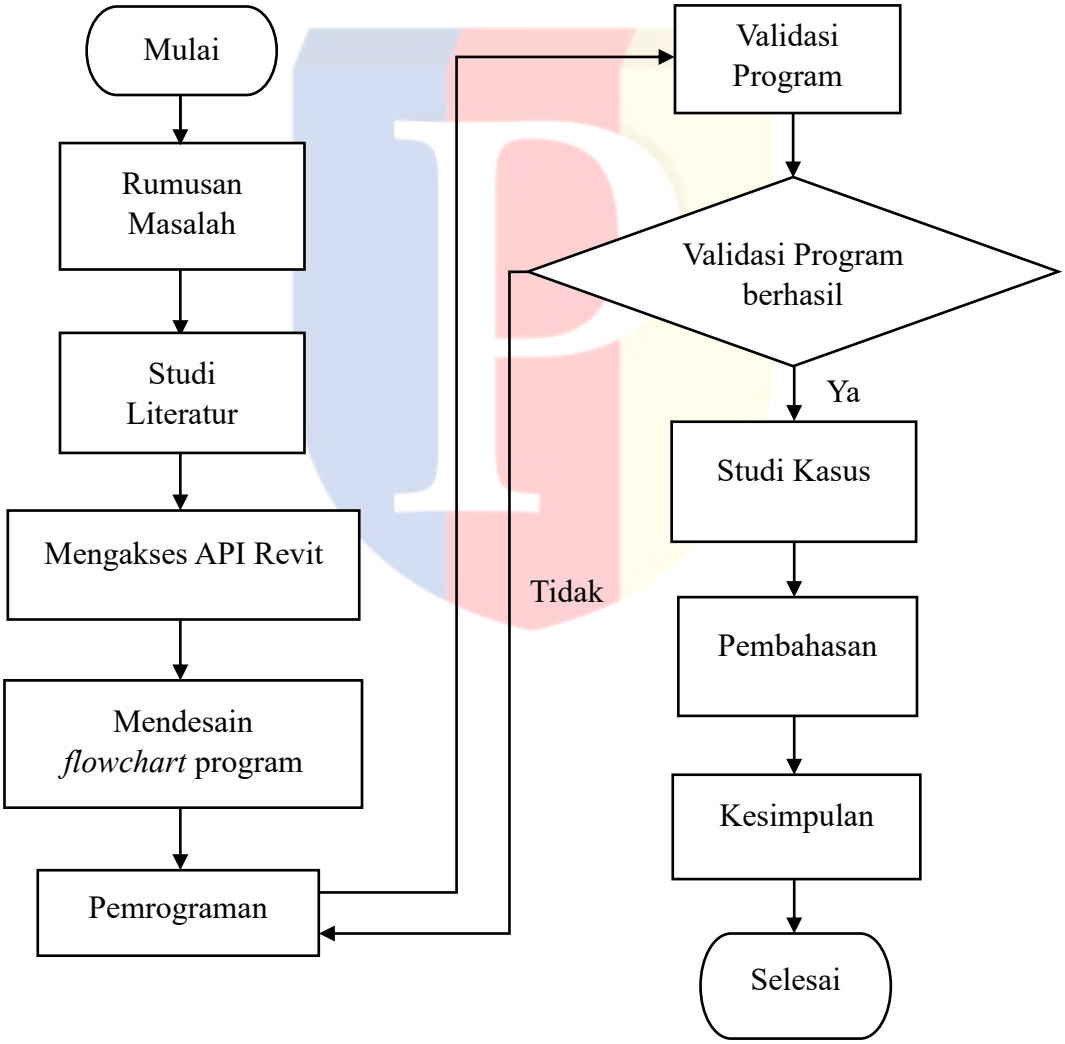
Tugas akhir "Penggunaan Bahasa Pemrograman Python dan Dynamo untuk Memodifikasi Perhitungan Volume Baja Tulangan pada Revit" merupakan laporan tertulis yang disusun untuk memodifikasi kemampuan *software* Revit dalam menghitung *overlapping* tulangan pada beton bertulang dan juga memerintahkan program untuk mengotomatisasi penggambaran tulangan pada Revit. Modifikasi pada *software* Revit akan menggunakan bantuan bahasa pemrograman Python untuk mengolah perhitungan pada tulangan, sehingga metode penelitian yang digunakan adalah metode simulasi. Menurut White dan Ingalls (2009), simulasi adalah pendekatan khusus untuk mempelajari model yang bersifat eksperiensial atau eksperimental. Hasil akhir dari tugas akhir ini diharapkan agar Revit dapat menghitung volume kebutuhan baja tulangan pada suatu struktur beton bertulang, dengan memperhitungkan faktor *overlapping* yang ada.

## III.2. Langkah-langkah Penelitian

Penelitian diawali dengan merumuskan masalah yang terjadi pada industri konstruksi, yaitu keterbatasan *software* Revit dalam menghitung dan memodelkan *overlapping* pada baja tulangan, yang mana keterbatasan tersebut akan berdampak pada perhitungan BoQ dan akan memengaruhi kebutuhan volume tulangan pada proyek. Studi literatur dilakukan untuk mengumpulkan sumber-sumber referensi terkait dengan masalah penelitian, yang mana masalah keterbatasan Revit dapat diatasi dengan melakukan modifikasi pada *software* dengan mengakses API Revit. Modifikasi tersebut akan berbasis pada penggunaan Bahasa Pemrograman Python dalam penulisan kode-kode pemrograman yang akan memungkinkan Revit untuk memperhitungkan faktor *overlapping* pada beton bertulang.

Langkah selanjutnya setelah melakukan studi literatur adalah mengakses API Revit dan melakukan simulasi mini pada Revit menggunakan Dynamo BIM. Simulasi mini ini dilakukan untuk mematangkan pemahaman mengenai penggunaan API pada Revit. Setelah penggunaan API telah matang, maka dilanjutkan dengan mendesain *flowchart* program sebagai alur kerja program ketika

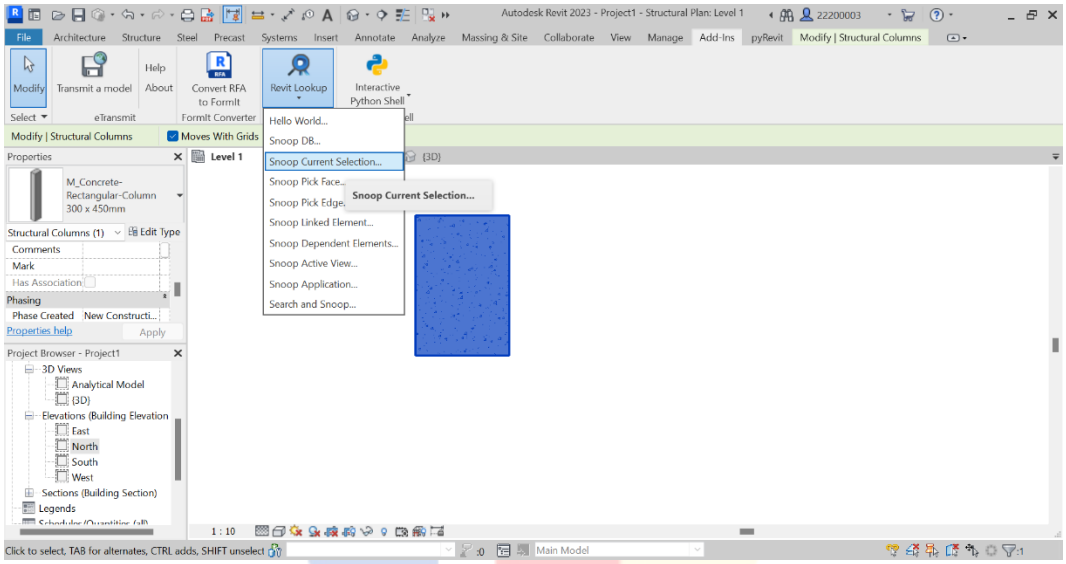
dijalankan pada Revit. Pemrograman dilakukan setelah *flowchart* program sudah final. Untuk memastikan bahwa penyusunan kode pemrograman sudah benar, maka dilakukan validasi program dengan menerapkannya pada suatu struktur portal sederhana. Jika validasi program belum berhasil, maka penelitian akan diulang ke tahap pemrograman untuk perbaikan. Namun, jika validasi program telah berhasil, maka program akan disiapkan untuk studi kasus yang menggunakan struktur hipotetik gedung 4 lantai. Setelah itu, akan dilanjutkan dengan melakukan pembahasan terkait temuan-temuan pada studi kasus, kemudian menyimpulkan hasil-hasil dari program yang menandakan tahap akhir dari penelitian. *Flowchart* penelitian dapat dilihat pada Gambar 3.1.



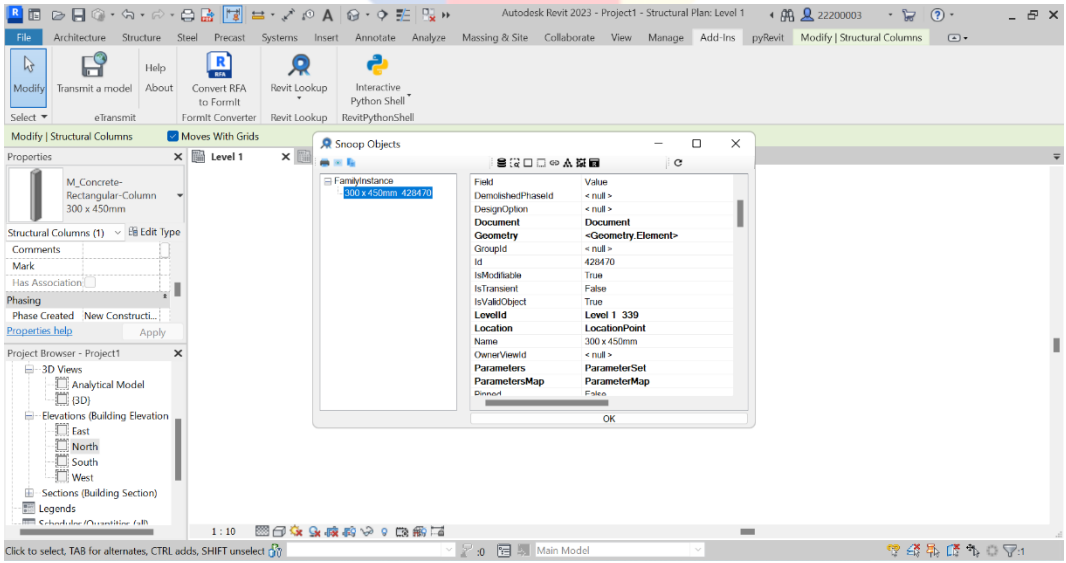

Gambar 3.1. *Flowchart* penelitian

### III.3. Mengakses API Revit

Akses API pada Revit menggunakan suatu fitur *add-in* bernama Revit Lookup. Untuk dapat mengakses API suatu elemen pada Revit, maka harus dibuat terlebih dahulu elemennya (misal: kolom). Kemudian klik pada elemen yang diinginkan menggunakan Revit Lookup supaya dapat diakses informasi-informasi mengenai elemen tersebut yang mana nantinya informasi tersebut akan digunakan pada saat proses pembuatan program. Cara menampilkan API dan tampilan informasi API pada Revit dapat dilihat pada Gambar 3.2 dan Gambar 3.3.

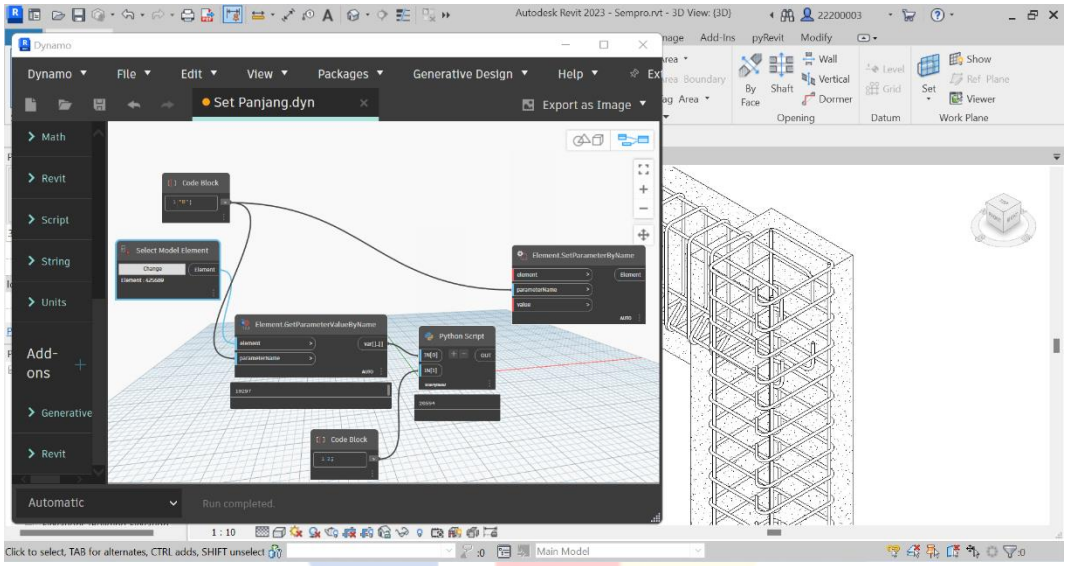



Gambar 3.2. Cara menampilkan API elemen pada Revit

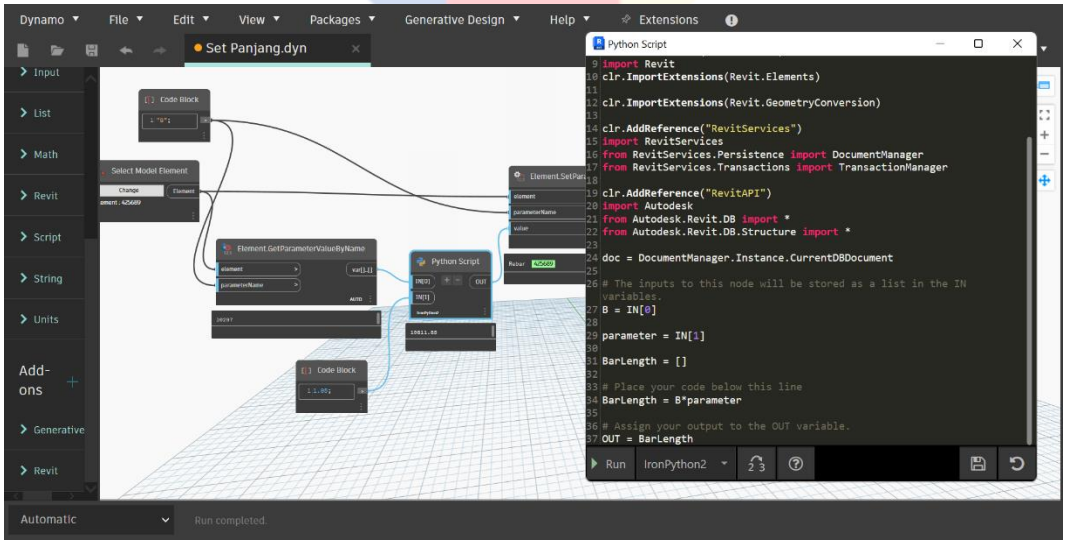



Gambar 3.3. Tampilan informasi API pada suatu elemen

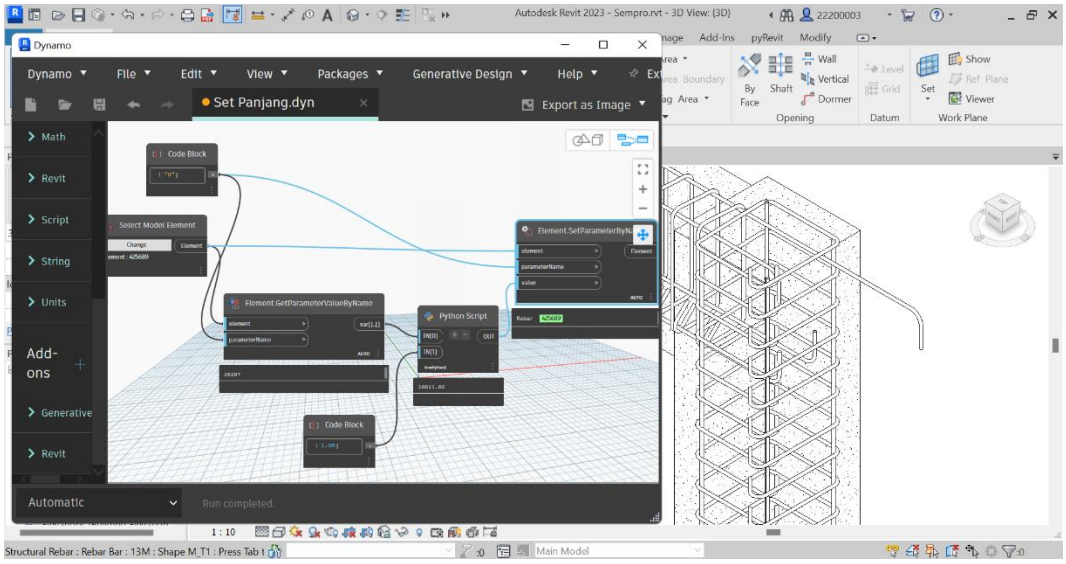
Gambar 3.4 merupakan contoh simulasi dalam memodifikasi parameter pada elemen Revit menggunakan Dynamo. Elemen yang akan dimodifikasi adalah tulangan baja, dan parameter yang akan dimodifikasi adalah panjangnya. Pada Dynamo, dibuat *node* sedemikian rupa untuk: menyeleksi elemen, memanggil nama parameter, menampilkan nilai elemen, dan mengatur nilai parameter elemen. Untuk melakukan modifikasi dengan Bahasa Pemrograman Python, dapat dilakukan dengan menambahkan *node Python Script* lalu menuliskan kode-kode pemrograman seperti pada Gambar 3.5.




Gambar 3.4. Tampilan dari Dynamo dan Revit




Gambar 3.5. Bentuk *Python Script* pada Dynamo

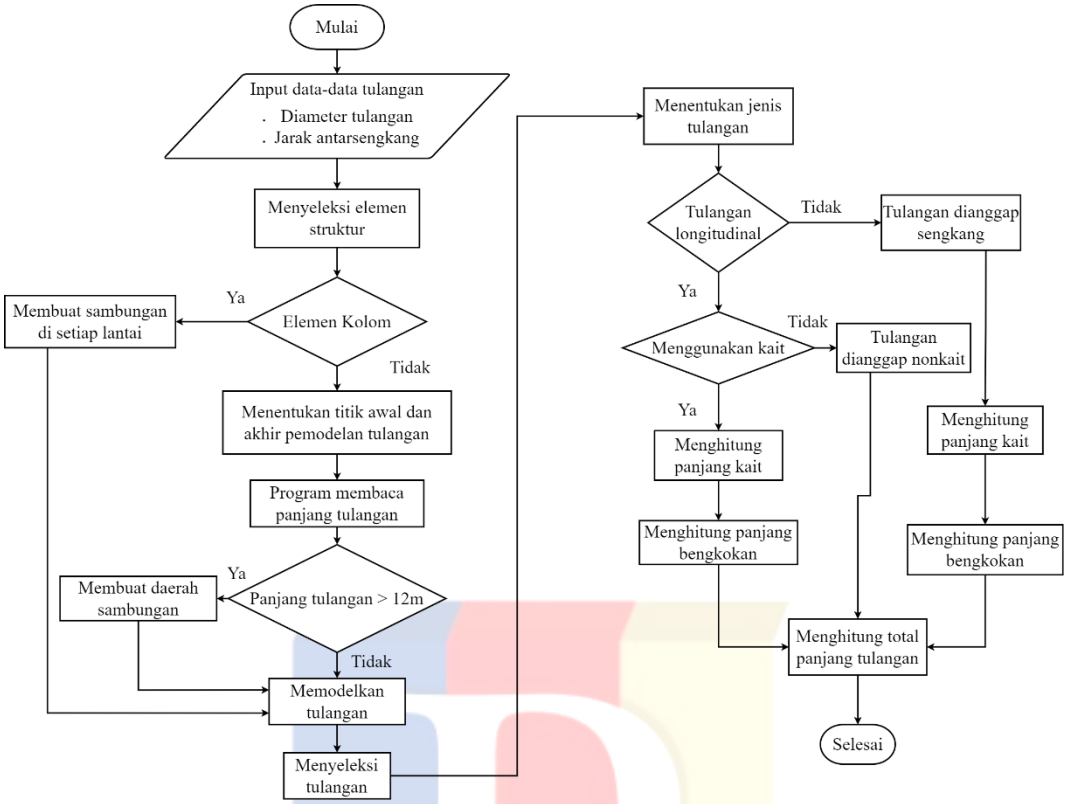



Gambar 3.6. Hasil modifikasi tulangan pada Revit

Pada *Python Script* disimulasikan agar nilai elemen yang dipilih dikalikan dengan angka yang diinginkan untuk mengubah panjang tulangan. Oleh karena itu, yang menjadi *input* pada *Python Script* adalah nama parameter yang ingin diubah dan besar nilai untuk mengubah nilai parameter. Sedangkan *output* berupa informasi mengenai hasil perkalian tersebut. Kemudian *output* tersebut dihubungkan pada *node* yang berfungsi dalam mengatur parameter, sehingga modifikasi pada elemen tersebut dapat dijalankan. Pada simulasi, modifikasi salah satu tulangan baja berhasil dilakukan. Hal ini dapat dilihat dengan membandingkan Gambar 3.4 dengan Gambar 3.6.

## III.4. *Flowchart* Program

Gambar 3.7 merupakan gambar alur kerja program. Program diawali ketika pengguna memasukkan data-data tulangan ke program, kemudian program akan mulai menyeleksi elemen struktur sebagai tempat tulangan akan dibuat. Program akan menentukan terlebih dahulu titik awal dan titik akhir pembuatan tulangan dengan menghitung panjang penyaluran ($l_d$). Di tahap ini, program juga telah menghitung panjang bentang tulangannya. Setelah itu, jika program membaca panjang tulangannya melebihi 12 m, maka program akan memodelkan tulangan disertai daerah sambungannya. Namun, jika elemen struktur yang terbaca adalah kolom, maka program daerah sambungan akan dimodelkan di setiap lantai gedung.

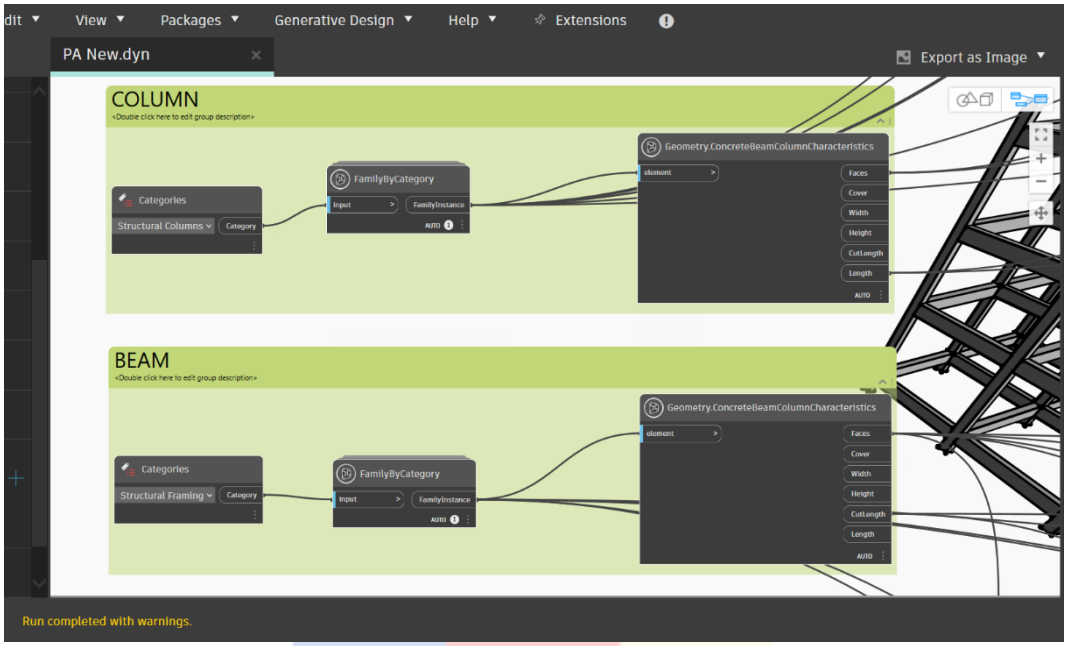



Gambar 3.7. *Flowchart* program

Setelah tulangan berhasil dimodelkan, maka program akan menyeleksi tulangan, lalu menentukan jenis tulangan tersebut (longituidinal atau sengkang). Apabila tulangan dibaca sebagai longitudinal, maka program akan membaca tulangan tersebut menggunakan kait atau tidak. Jika tulangan menggunakan kait, maka program akan menghitung panjang kaitnya, kemudian panjang bengkokannya, lalu total panjang tulangannya. Namun, jika tulangan tidak menggunakan kait, maka program akan langsung menghitung total panjangnya. Apabila program membaca tulangan sebagai sengkang, maka proses perhitungannya akan sama seperti proses ketika menghitung tulangan kait.

### III.5. Implementasi Program

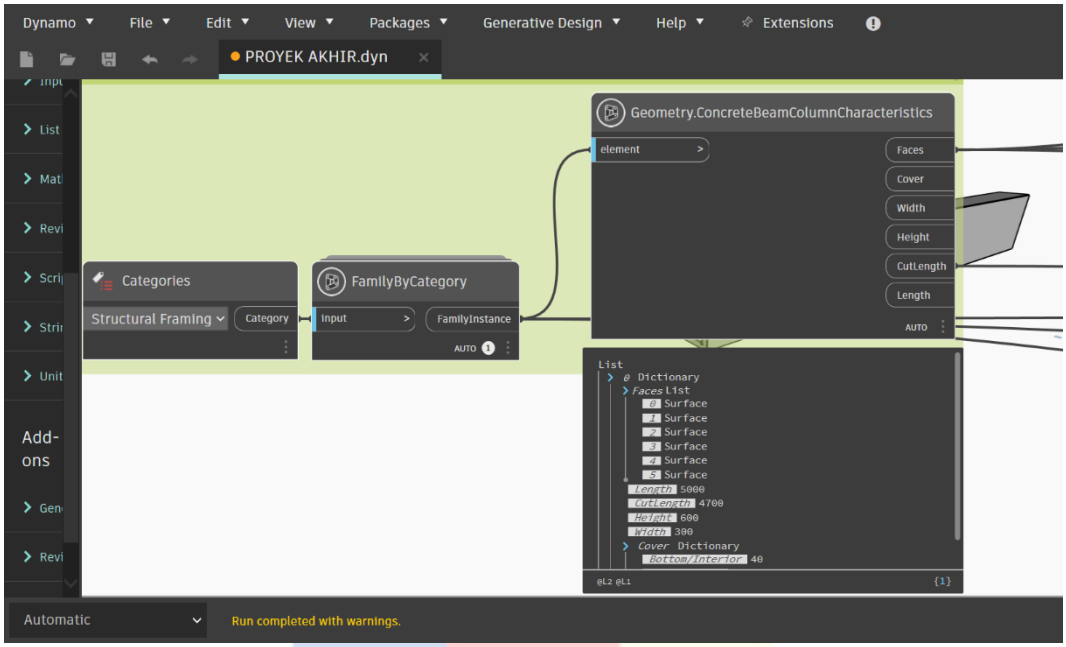
Implementasi program berdasarkan *flowchart* pada Gambar 3.7 dengan mengambil contoh kasus struktur portal sederhana yang terdiri dari 2 (dua) elemen kolom dan 1 (satu) elemen balok.



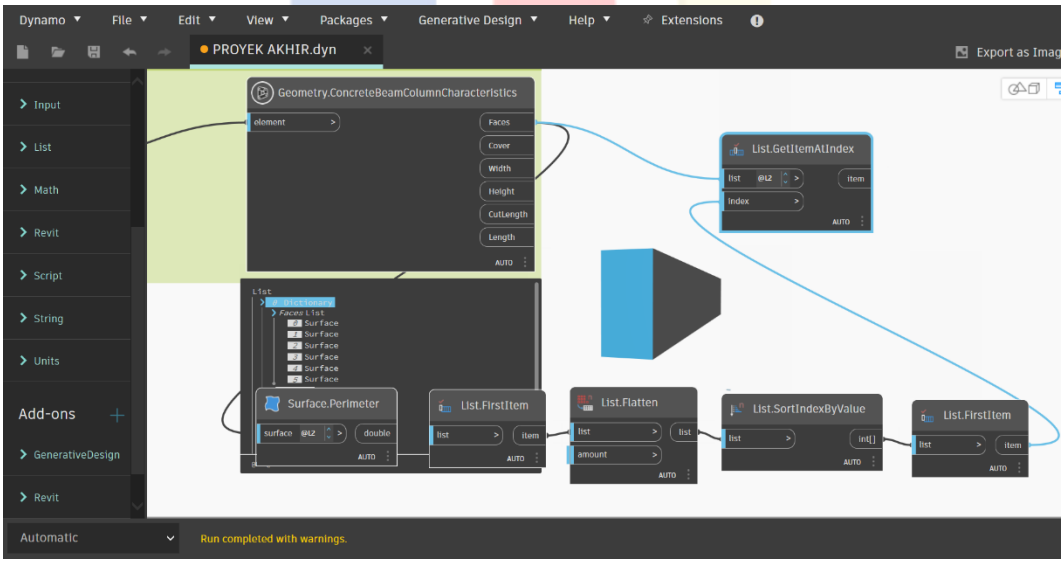

Gambar 3.8. *Nodes* untuk menetapkan kategori struktur

Pada bagian desain, program dijalankan menggunakan pemrograman visual Dynamo yang melibatkan *node* dalam menjalankan program. Gambar 3.8 menunjukkan dua kelompok (*Group*) besar untuk kategori kolom dan balok. Pada masing-masing *Group* terdiri dari 3 *nodes* dengan alur kerja:

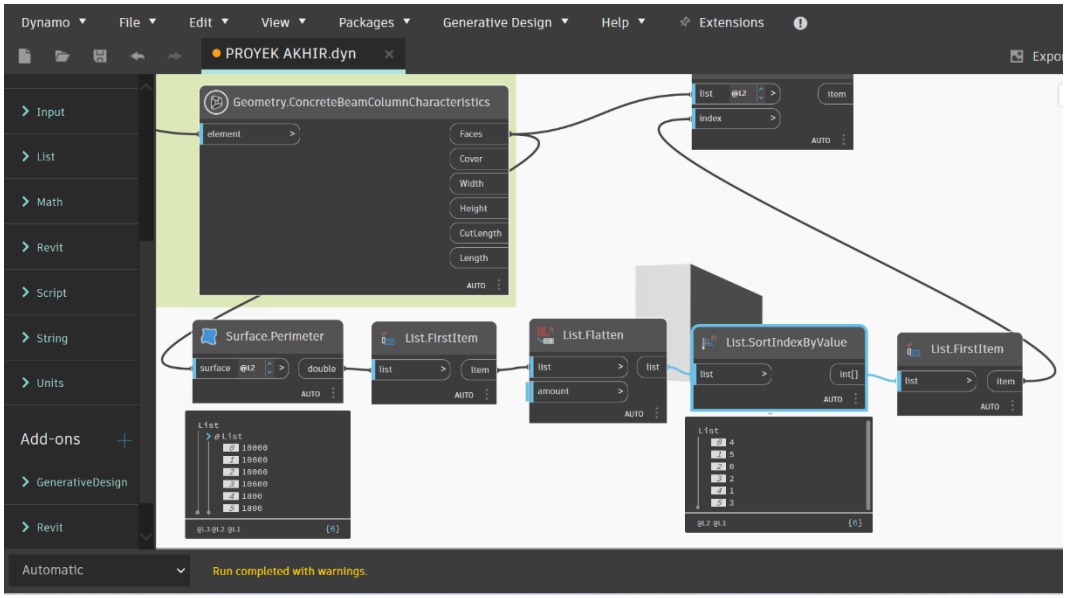
1. *Categories*, menetapkan kategori struktur yang akan diseleksi, misal: *Structural Columns* untuk kolom dan *Structural Framing* untuk balok;
2. *FamilyByCategory*, menghitung banyaknya elemen struktur yang telah ditentukan pada *node Categories*, kemudian menyatakan elemen-elemen tersebut dalam bentuk ID;
3. *Geometry.ConcreteBeamColumnCharacteristics*, menampilkan informasi mengenai elemen pada *node FamilyByCategory* dan memungkinkan informasi tersebut untuk dijadikan *input* pada *node* selanjutnya. Gambar 3.9 merupakan contoh informasi yang ditampilkan oleh *node* ini. Untuk balok, informasi elemen yang akan digunakan untuk *node* selanjutnya adalah informasi *Faces* (menunjukkan banyaknya permukaan yang dimiliki elemen balok) dan *CutLength* (menunjukkan panjang bersih tiap bentang balok). Sedangkan untuk kolom menggunakan informasi *Faces* dan *Length* (menunjukkan tinggi kolom).




Gambar 3.9. *Node Geometry.ConcreteBeamColumnCharacteristics* pada elemen balok



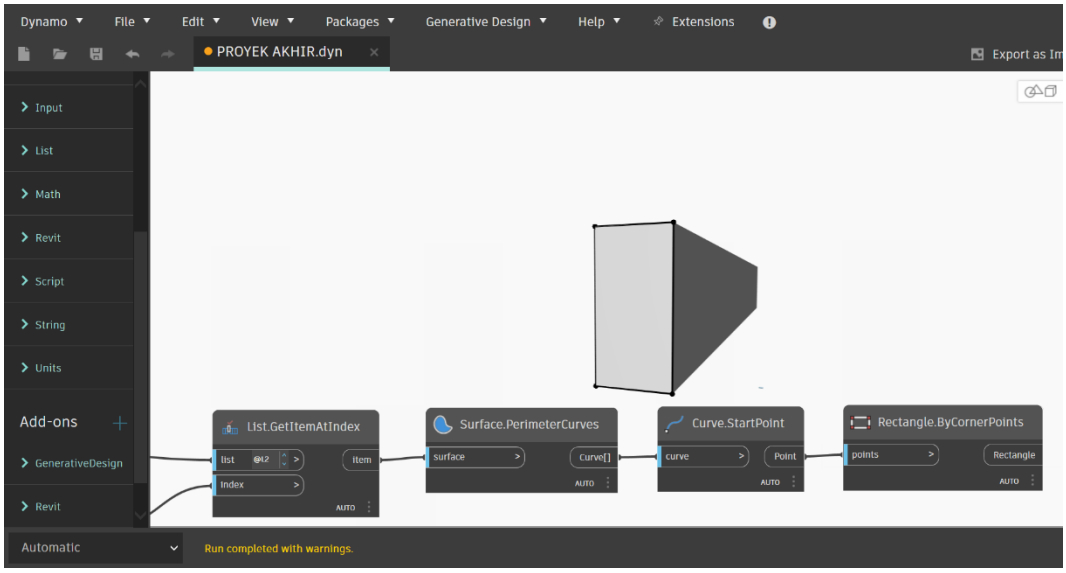

Gambar 3.10. Menentukan posisi awal untuk tulangan




Gambar 3.11. Mengurutkan nilai keliling permukaan pada balok

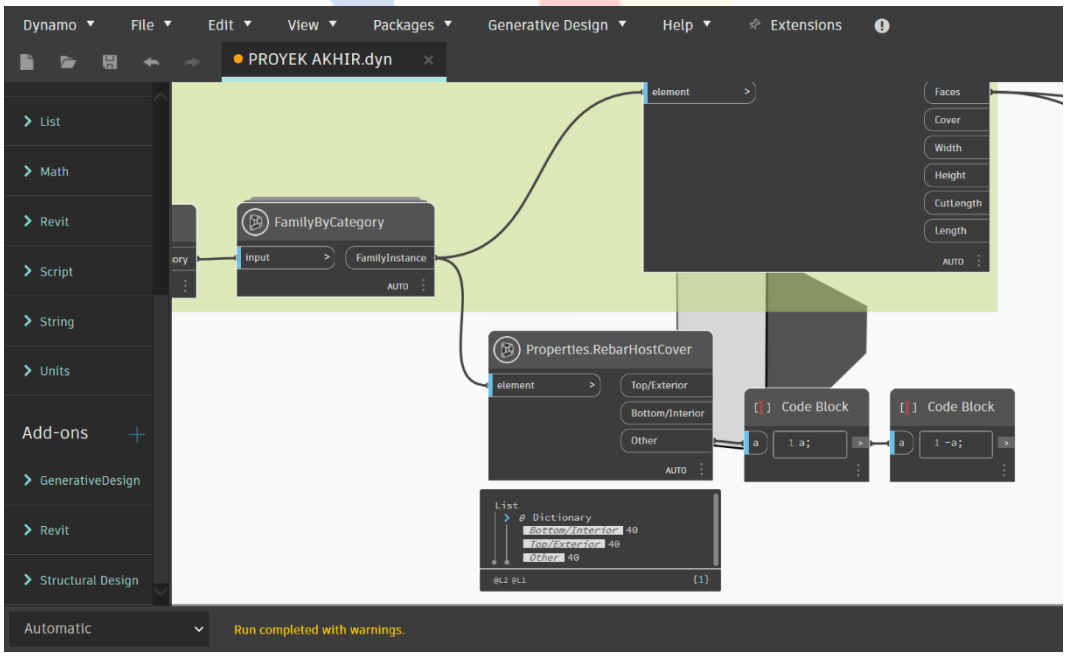
Dalam membuat tulangan pada elemen struktur, maka harus ditentukan posisi mulai dan akhir dari tulangan tersebut, baik tulangan longitudinal maupun tulangan sengkang. Gambar 3.10 menunjukkan bahwa *node List.GetItemAtIndex* sedang menyeleksi permukaan penampang balok sebagai posisi mulai dari penempatan tulangan. Agar program dapat menyeleksi permukaan seperti yang terlihat pada Gambar 3.10 (permukaan yang berwarna biru), maka harus dirangkai *nodes* seperti pada Gambar 3.11.

*Node Surface.Perimeter* berfungsi untuk menghitung keliling permukaan pada suatu elemen. Kasus pada Gambar 3.11 menggunakan elemen balok, dan balok memiliki 6 (enam) permukaan, sehingga *node* menampilkan enam nilai ([0]10.000, [1]10.600, [2]10.000, [3]10.600, [4]1.800, dan [5]1.800). Setelah itu, keenam nilai tersebut diurutkan menggunakan *node List.SortIndexByValue* dengan urutan: [4], [5], [0], [1], [2], dan [3]. Kemudian *node List.FirstItem* berfungsi untuk hanya mengambil nilai pertama dari *node* sebelumnya, yaitu [4]. Hasil dari *node List.FirstItem* menjadi input bagi *node List.GetItemAtIndex*, sehingga *node* tersebut hanya menyeleksi permukaan [4] sebagai posisi mulai dari tulangan.



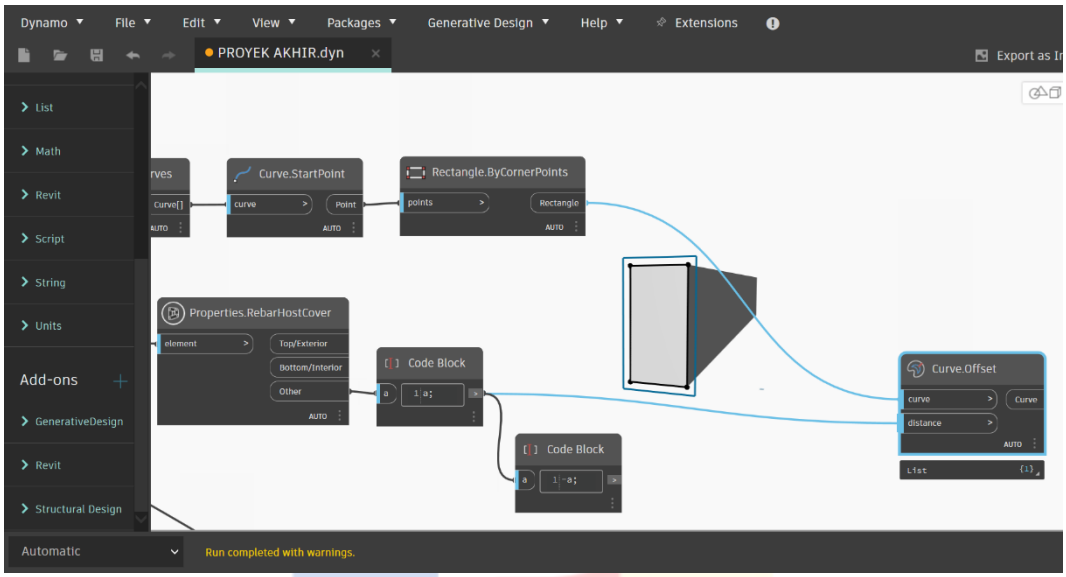

Gambar 3.12. Membuat kurva pada permukaan posisi mulai

Tulangan pada beton harus dipasang sesuai jarak selimut betonnya. Dalam membuat tulangan pada selimut beton, maka langkah yang harus dilakukan adalah membuat kurva pada keliling permukaan elemen yang telah dipilih. *Node Surface.PerimeterCurves* berfungsi dalam membuat kurva tersebut. Lalu, *node Curve.StartPoint* secara otomatis membuat titik pada sudut-sudut kurva yang telah terbentuk. Kemudian *node Rectangle.ByCornerPoints* akan membuat objek persegi dari titik-titik tersebut.

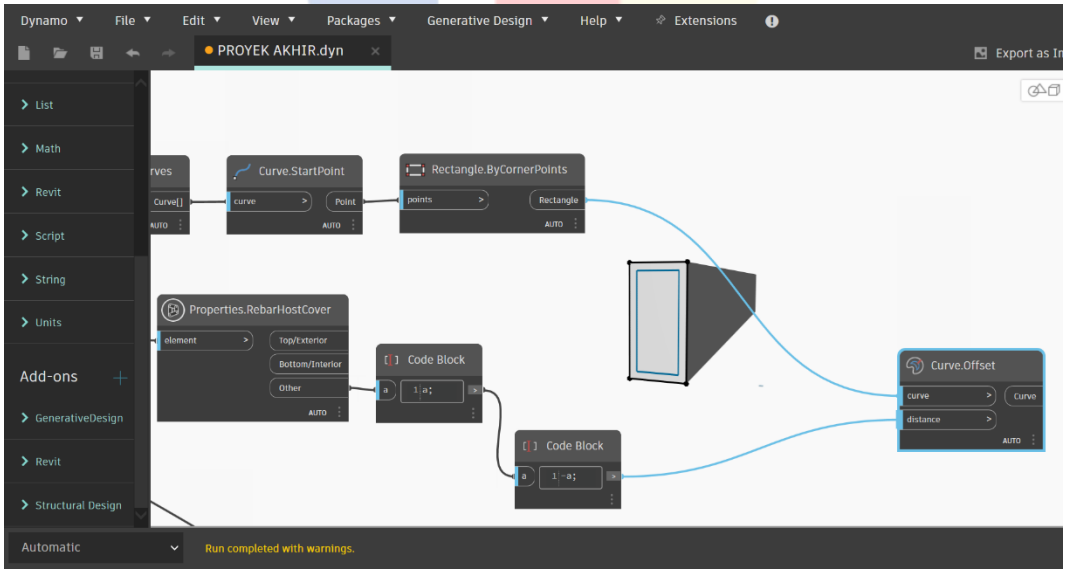



Gambar 3.13. *Node* menentukan selimut beton

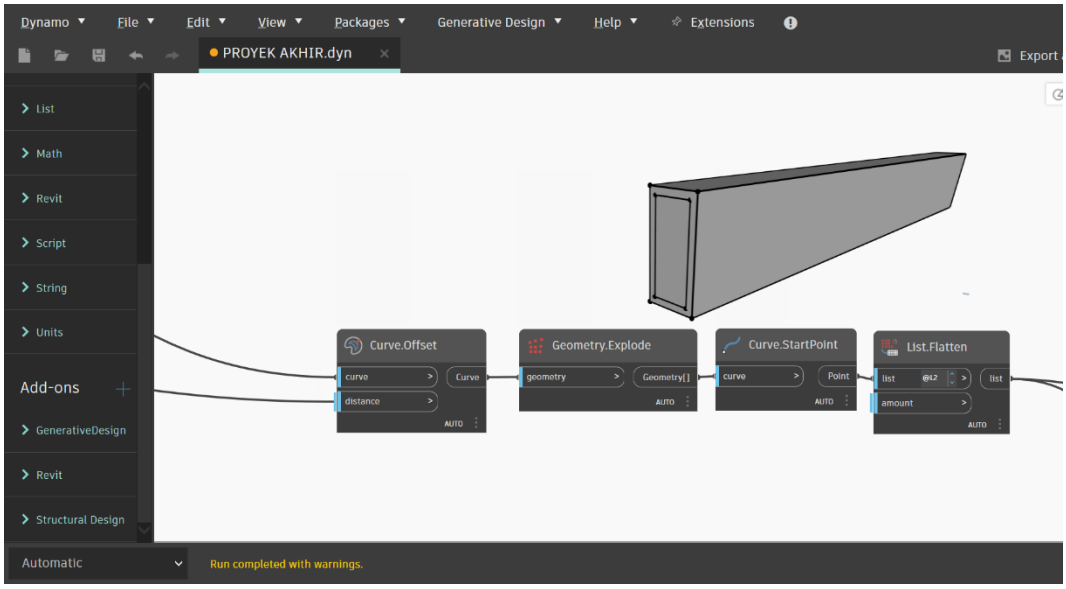
*Node Properties.RebarHostCover* pada Gambar 3.13 akan menampilkan informasi mengenai ukuran selimut beton pada elemen yang terbaca pada *node FamilyByCategory*. Nilai selimut beton pada *node Properties.RebarHostCover* harus bernilai negatif supaya garis *offset* selimut beton berada di dalam elemen. Gambar 3.14 dan Gambar 3.15 dapat membantu dalam memahami pengaruh nilai negatif terhadap garis *offset* selimut beton.




Gambar 3.14. Bentuk garis *offset* selimut beton jika *node* bernilai positif

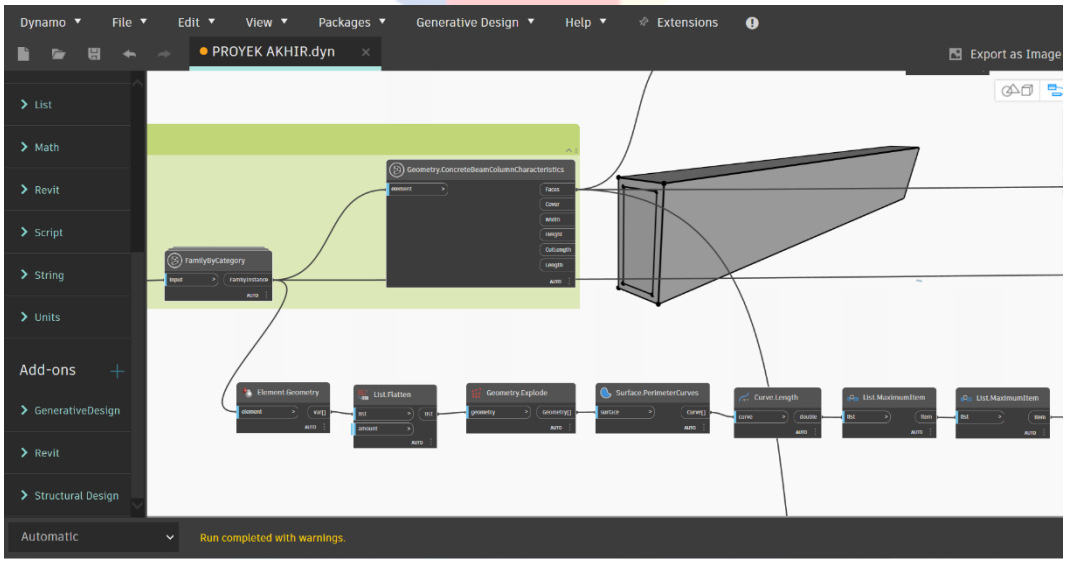



Gambar 3.15. Bentuk garis *offset* selimut beton jika *node* bernilai negatif



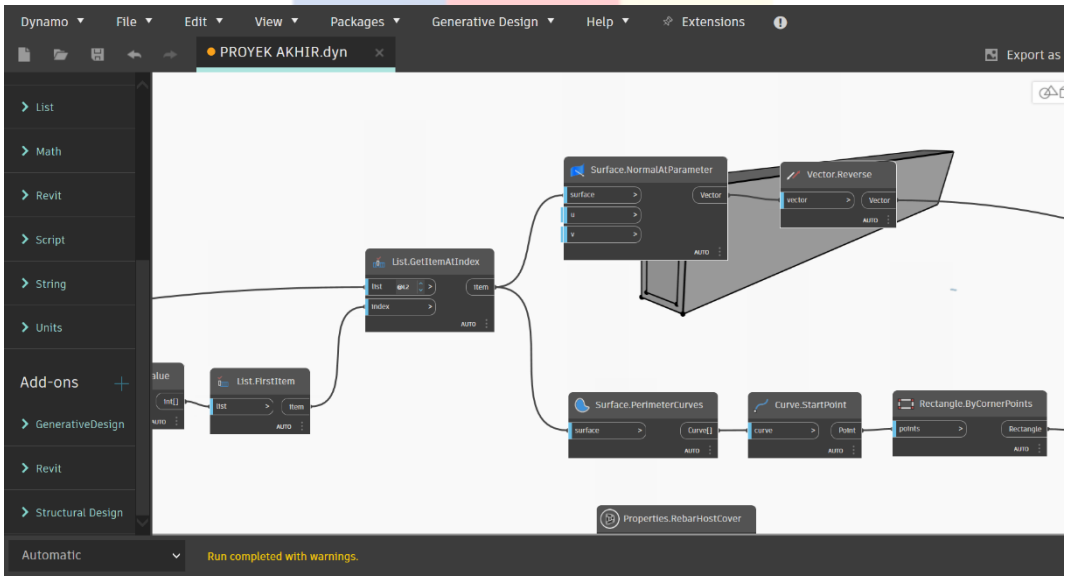

Gambar 3.16. Menentukan titik awal tulangan longitudinal

Tulangan longitudinal harus terpasang pada area selimut beton. Maka dari itu, garis *offset* selimut beton pada Gambar 3.16 harus dipecah menggunakan *node Geometry.Explode*, kemudian dibuat titik pada sudut-sudut selimut beton menggunakan *node Curve.StartPoint*. *Node List.Flatten* berfungsi dalam mengambil nilai dari *node* sebelumnya dan mengumpulkan nilai tersebut dalam bentuk *list*, dengan keadaan jika terdapat lebih dari satu elemen balok. Namun, karena contoh ini hanya menggunakan satu balok, sehingga *node List.Flatten* hanya meneruskan nilai dari *node Curve.StartPoint*.



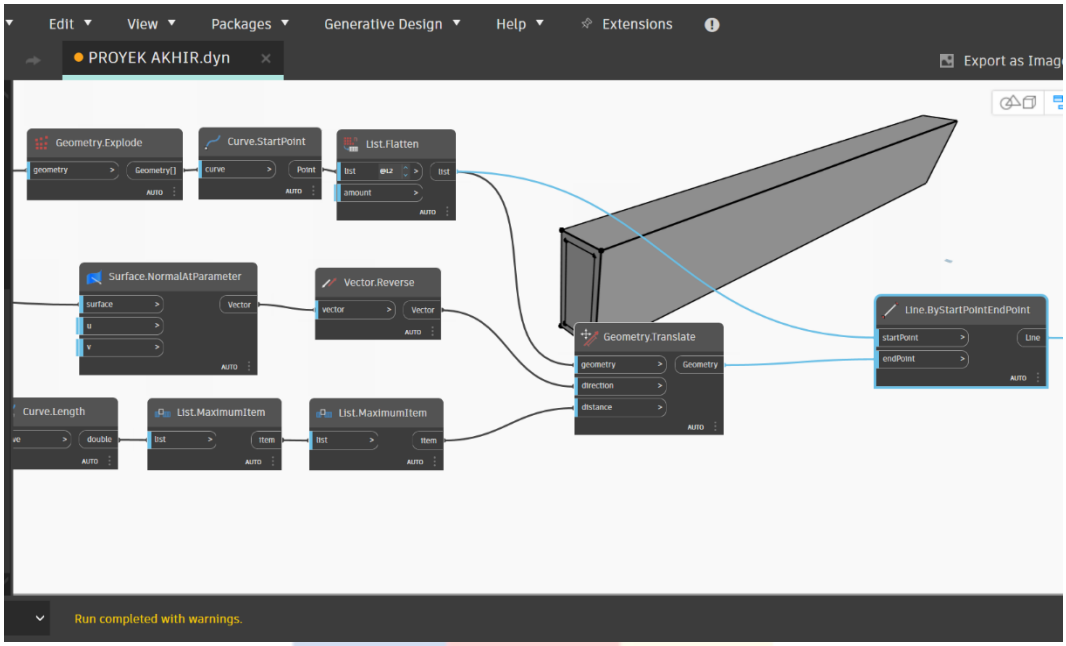

Gambar 3.17. Menentukan jarak titik akhir tulangan longitudinal

Salah satu *input* dalam menentukan titik akhir tulangan longitudinal adalah menentukan jarak titik akhir tersebut dari titik awal, dan arah vektor dari titik akhir tersebut. Panjang tulangan longitudinal adalah sepanjang bentang bersih balok. Maka dari itu, elemen balok harus diterjemahkan ke dalam bentuk geometri dengan *node Element.Geometry* (Gambar 3.17). Hasil geometri harus dipecah dengan *node Geometry.Explode* menjadi 6 permukaan, kemudian menerjemahkan keenam permukaan tersebut ke bentuk kurva dengan *node Surface.PerimeterCurves,* serta membaca panjang kurva tersebut dengan *node Curve.Length*. Hasil dari *Curve.Length* adalah enam *list* permukaan balok beserta panjangnya. *Node List.MaximumItem* akan mengambil nilai maksimal dari *node* sebelumnya. Oleh karena itu, *node List.MaximumItem* akan menentukan panjangnya jarak titik akhir dari tulangan.



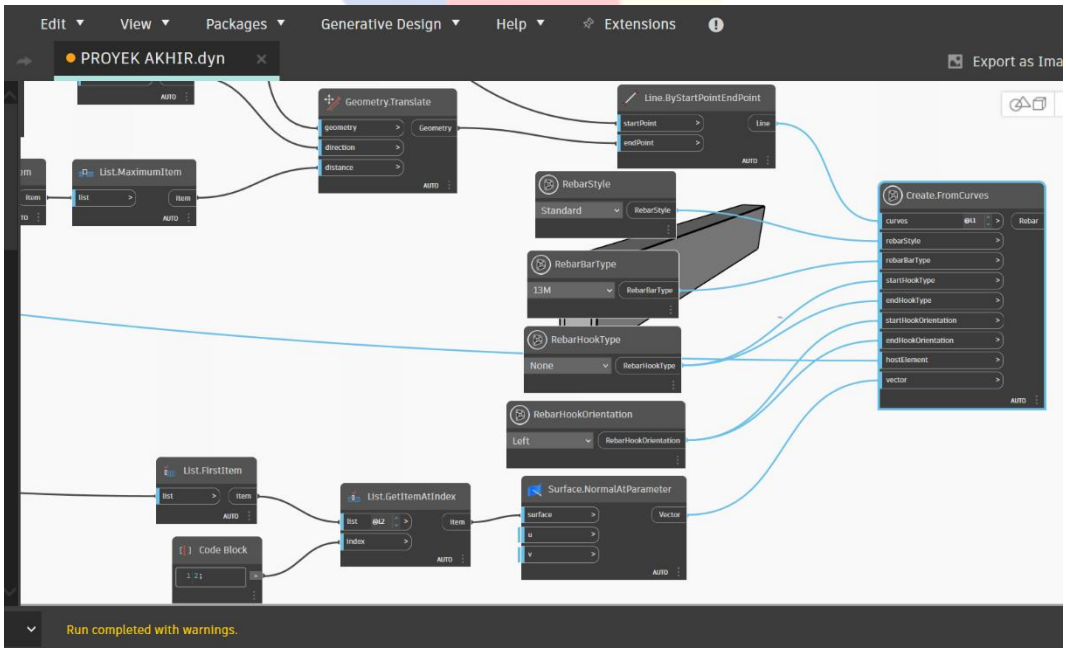

Gambar 3.18. Menentukan arah vektor titik akhir tulangan longitudinal

Pada Gambar 3.18, *node Surface.NormalAtParameter* berfungsi dalam mengubah permukaan elemen ke bentuk vektor. Namun, nilai vektor pada *node* tersebut akan bernilai negatif. Jika vektornya bernilai negatif, maka tulangan akan terpasang ke arah yang berlawanan dari elemen balok. Oleh karena itu, *node Vector.Reverse* akan diubah menjadi arah yang berlawanan, yaitu positif.




Gambar 3.19. Menetapkan titik awal dan akhir dari tulangan longitudinal

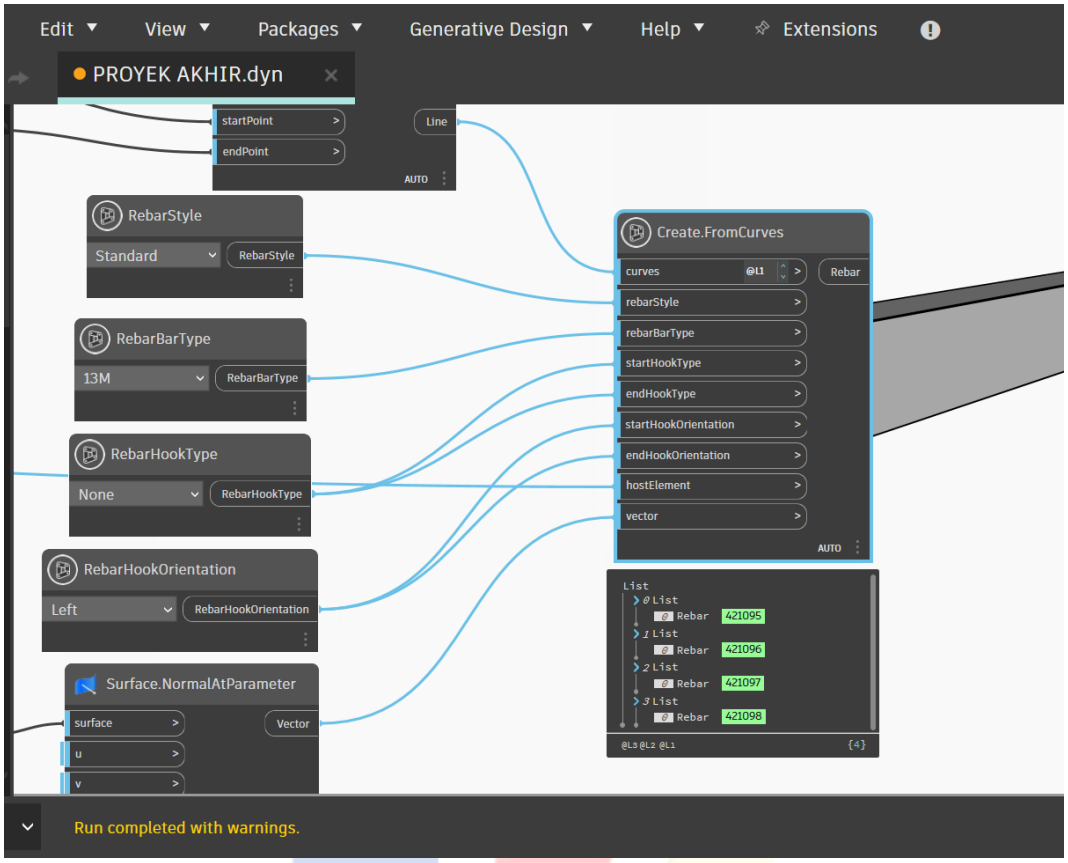
*Node Geometry.Translate* berfungsi dalam menerjemahkan bentuk-bentuk geometri sesuai dengan arah vektor dan jarak yang telah ditentukan. *Input* geometri adalah *node List.Flatten* yang berisi posisi koordinat titik awal tulangan. Pada Gambar 3.19, *Node Geometry.Translate* menerjemahkan titik awal menjadi titik akhir tulangan. Sehingga kedua titik tersebut akan menjadi *input* bagi *node Line.ByStartPointEndPoint* dan menghasilkan *output* berupa garis kurva.




Gambar 3.20. Membuat tulangan

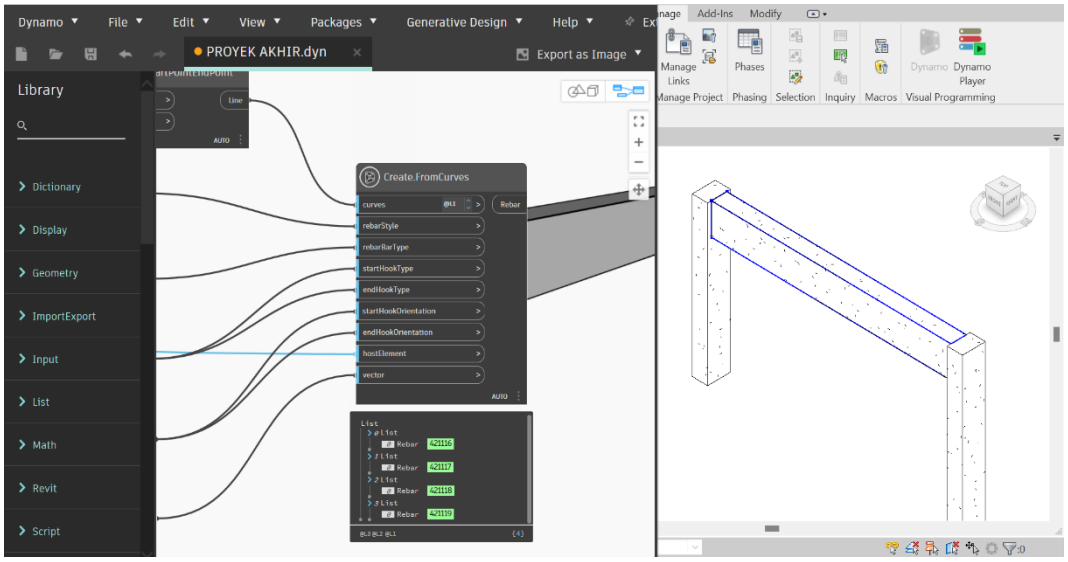
Pada Gambar 3.20, *node Create.FromCurves* akan membuat tulangan berdasarkan *input* berikut: *curves, rebarStyle, rebarBarType, startHookType, endHookType, startHookOrientation, endHookOrientation, hostElement,* dan *vector*. Berikut merupakan penjelasan mengenai *input* tersebut.

- *Input curves* diambil dari *node Line.ByStartPointEndPoint*, yang menjadikan garis kurva sebagai panjang tulangan longitudinal;
- *Input rebarStyle* dapat diambil dari *node RebarStyle*, yang memberikan pilihan jenis tulangan, yaitu: *Standard* (longitudinal) dan *StirrupTie* (sengkang);
- *Input rebarBarType* dapat diambil dari *node RebarBarType*, yang memberikan pilihan diameter tulangan;
- *Input startHookType* dan *endHookType* dapat diambil dari *node RebarHookType*, yang memberikan pilihan mengenai tipe bengkokan (kait) pada tulangan. Jika tulangan tidak ingin menggunakan kait, maka dapat memilih *None* pada *node*;
- *Input startHookOrientation* dan *endHookOrientation* dapat diambil dari *node RebarHookOrientation*, yang merupakan mengatur orientasi kait (ke kiri atau ke kanan);
- *Input hostElement* adalah *node FamilyByCategory* seperti pada Gambar 3.8. *hostElement* menegaskan bahwa tulangan harus terpasang di dalam elemen struktur (balok atau kolom); dan
- *Input vector* berasal dari *node Surface.NormalAtParameter* yang merupakan vektor dari permukaan memanjang balok.

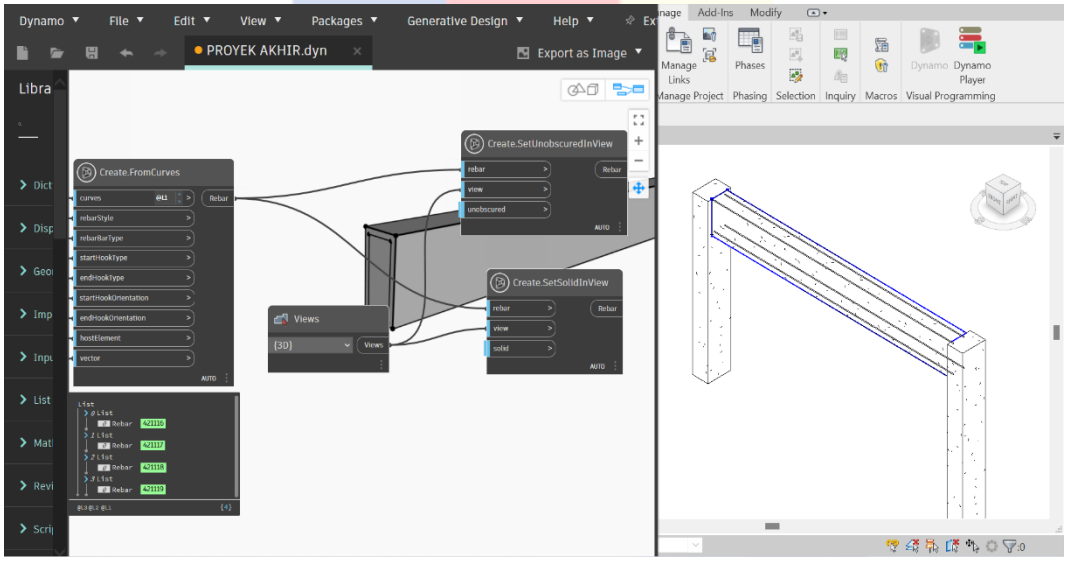



Gambar 3.21. Hasil tulangan dari *node Create.FromCurves*

Jika semua *input* pada *node Create.FormCurves* telah terpasang dengan benar, maka *node* akan membuat tulangan yang diinginkan. Gambar 3.21 menunjukkan bahwa *node* telah berhasil membuat empat tulangan, yang mana keempat tulangan tersebut ditampilkan dalam bentuk ID. Walaupun Dynamo berhasil membuat keempat tulangan tersebut, namun hasil tulangannya tidak akan terlihat pada Revit, seperti pada Gambar 3.22. Maka dari itu, perlu ditambahkan *nodes Create.SetUnobscuredInView* dan *Create.SetSolidInView* dengan menghubungkan *node Views*, seperti pada Gambar 3.23 untuk menampilkan tulangan pada Revit.



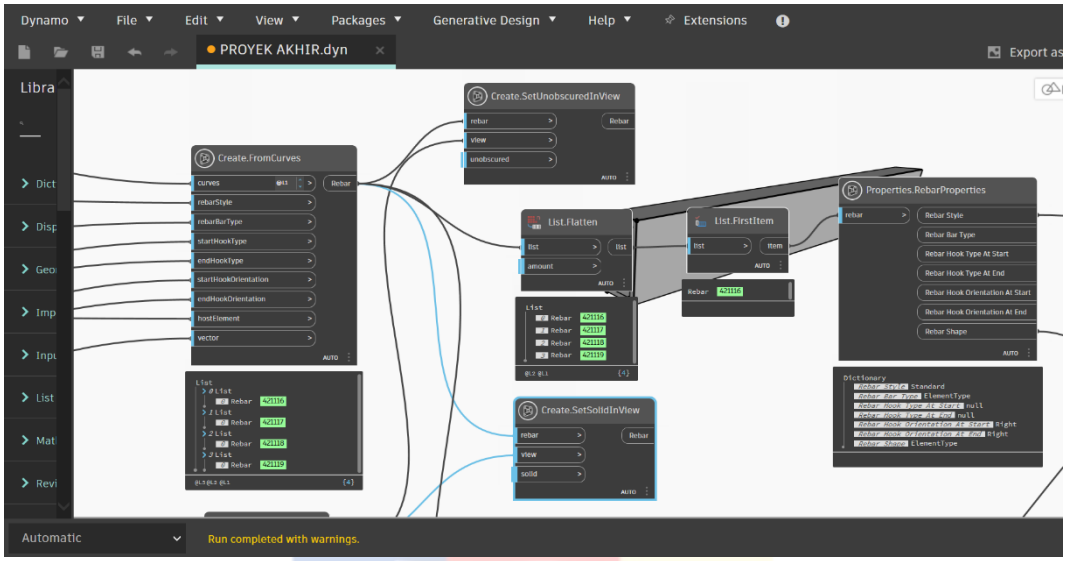

Gambar 3.22. Hasil *node Create.FromCurves* pada Revit



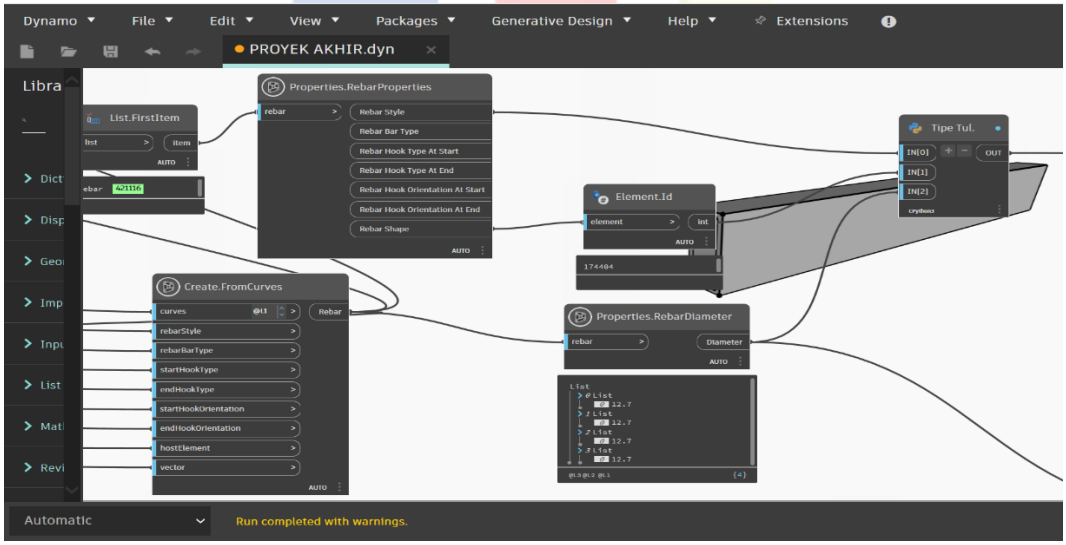

Gambar 3.23. Tampilan tulangan longitudinal pada Revit

Setelah tulangan berhasil dimodelkan pada Revit, maka program akan mulai menghitung panjang tulangan. *Node Create.FromCurves* menyimpan hasil tulangan dalam bentuk *list* (kumpulan objek). Oleh karena itu, untuk memudahkan dalam penulisan kode pemrograman, maka akan diambil salah satu tulangan dengan *nodes List.Flatten* dan *List.FirstItem*. Hasilnya dari keempat tulangan yang identik, hanya satu yang akan dimasukkan ke dalam perhitungan program. Pada Gambar 3.24, dapat dilihat bahwa *node List.Flatten* meratakan objek tulangan pada *node Create.FromCurves*, dan *node List.FirstItem* hanya mengambil objek pertama

dalam *list* pada *node List.Flatten.* Kemudian objek tersebut akan diteruskan pada *node Properties.RebarProperties*.




Gambar 3.24. Menampilkan properti tulangan

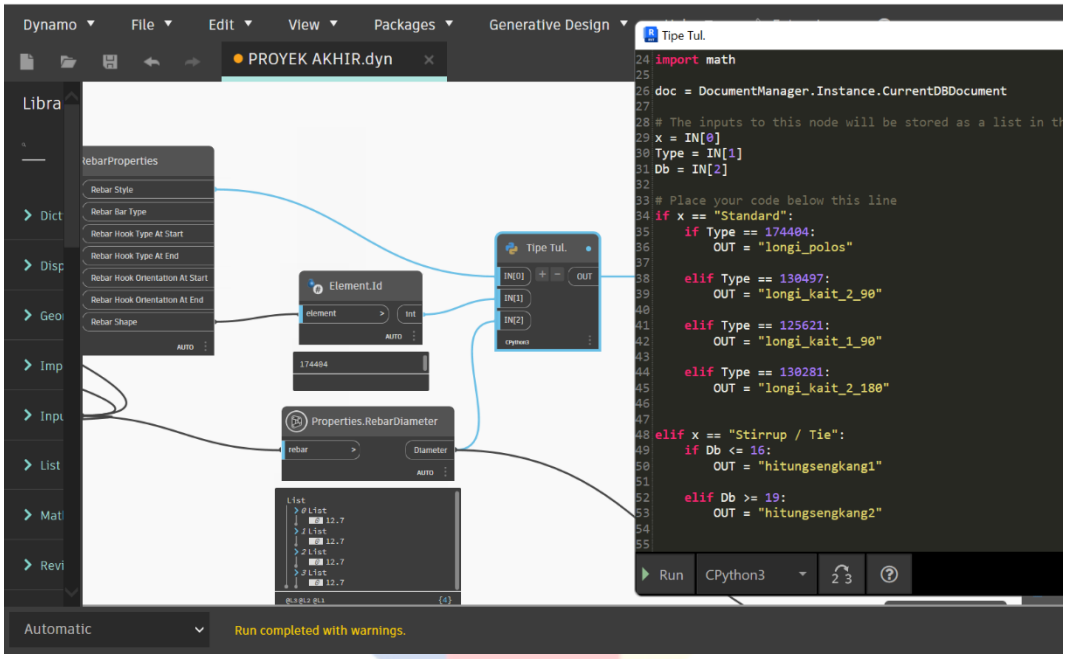



Gambar 3.25. *Input* pada *node PythonScript* (Tipe.Tul)

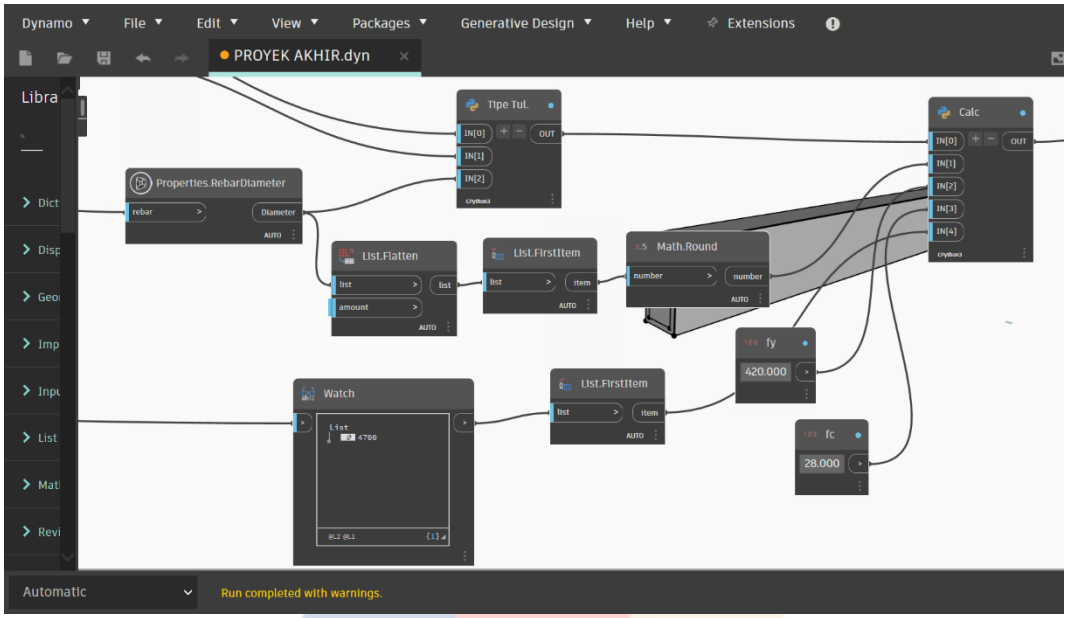
Pada Gambar 3.25., terdapat *node* bernama Tipe Tul. yang sebenarnya merupakan tipe *node PyhtonScript* yang memungkinkan pengguna untuk menuliskan atau memasukkan kode pemrograman Python pada Dynamo. Jika dilihat, maka *node* Tipe Tul. memiliki tiga *input* dengan rincian:

- IN[0] merupakan *Rebar Style* atau jenis tulangan, yaitu *Standard* (longitudinal) atau *StirrupTie* (sengkang);

- IN[1] merupakan *node Element.Id* yang mengambil informasi dari *Rebar Shape*. Jika dilihat, *node* ini menampilkan nilai 174404 yang merupakan kode untuk tulangan longitudinal polos. Kode-kode untuk jenis tulangan lainnya dapat dilihat pada Gambar 3.26;
- IN[2] merupakan *node Properties.RebarDiameter* yang mengambil informasi *Rebar* dari *node Create.FromCurves*. *Node Properties.RebarDiameter* akan menampilkan informasi mengenai diameter-diameter tulangan yang telah dibuat oleh program.




Gambar 3.26. Isi skrip Python pada *node* Tipe Tul.



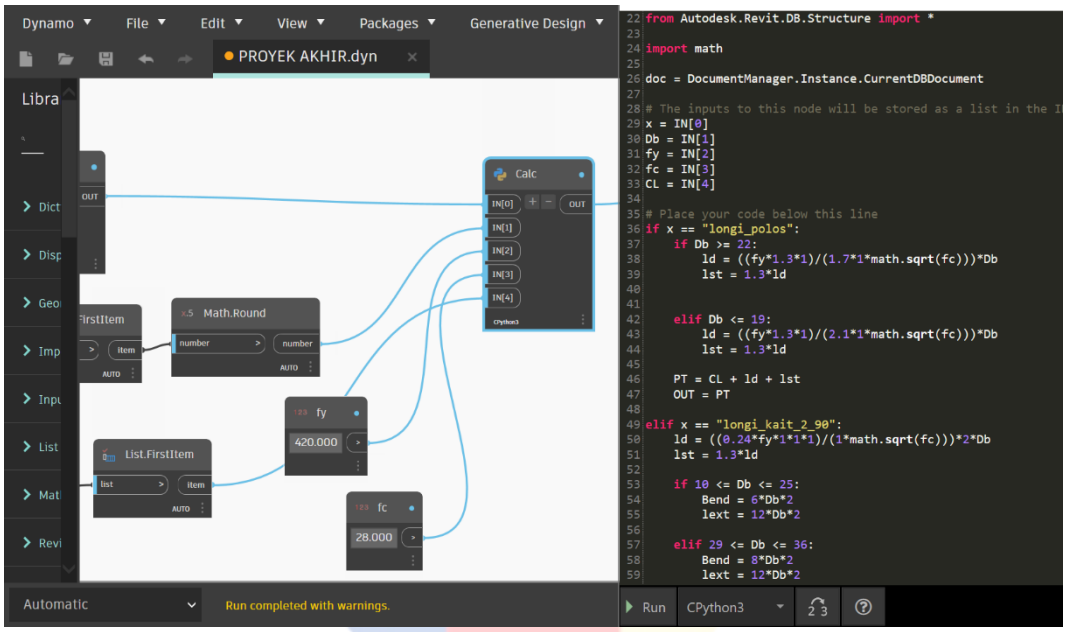

Gambar 3.27. *Input* pada *node Calc*

Gambar 3.27 menunjukkan bahwa *output* dari *node* Tipe Tul. (longi_polos) akan menjadi *input* bagi *node Calc*. *Node Calc* juga merupakan *node PythonScript* yang berisi perhitungan persamaan dalam menghitung panjang total tulangan beserta panjang penyaluran dan panjang sambungannya. Ketentuan mengenai persamaan panjang penyaluran ($l_d$) dan panjang sambungan ($l_{st}$) mengikuti subbab II.2 yang mengacu pada ketentuan SNI 2847:2019. Pada Gambar 3.28, dapat dilihat bahwa terdapat empat *input* pada *node Calc*, yaitu:

- IN[0] merupakan *output* dari *node* Tipe Tul. yang hasilnya adalah tipe tulangan (“longi_polos”, “longi_kait_2_90”, “longi_kait_1_90”, “longi_kait_2_180”). Dalam contoh ini, *output* dari *node* ini adalah “longi_polos”;
- IN[1] merupakan nilai diameter tulangan. Pada Gambar 3.25., dapat dilihat bahwa walaupun menggunakan tipe D13, namun Revit membaca diameter asli dari tulangan tersebut adalah 12,7. Oleh karena itu, nilai tersebut perlu

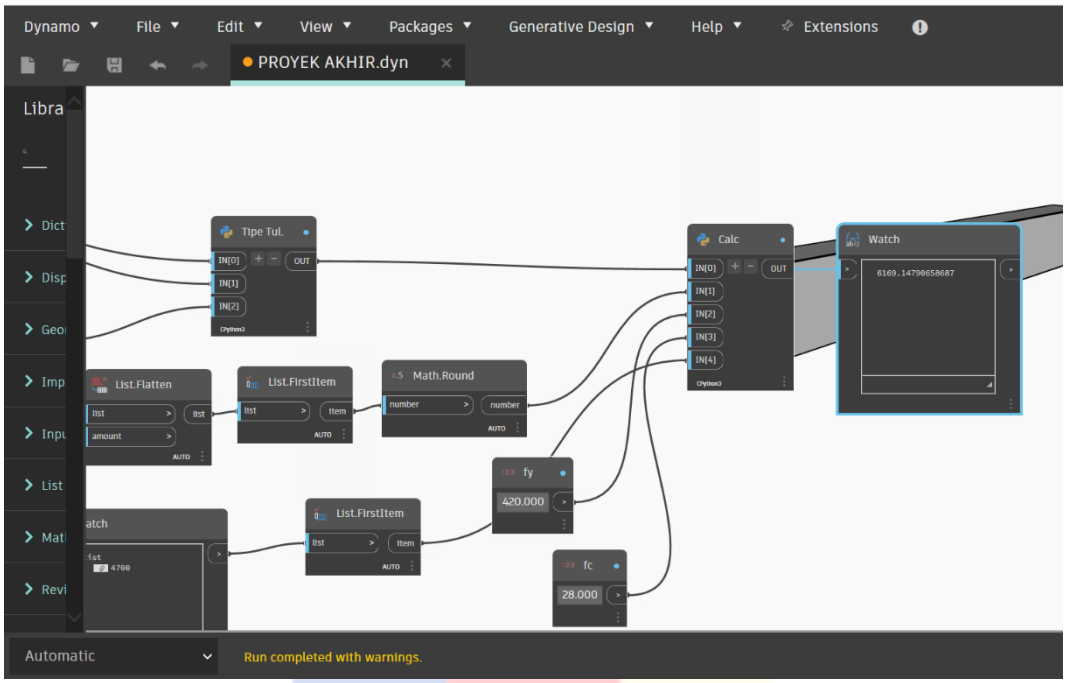
dibulatkan menggunakan *node Math.Round* sehingga diameter tulangan dibaca sebagai 13;

- IN[2] merupakan nilai $f_y$, yang mana nilai tersebut dapat ditulis menggunakan *node* bernama *Code Block*;
- IN[3] merupakan nilai $f_c$'. Sama seperti IN[2], nilai ini juga dapat ditulis menggunakan *node Code Block*;
- IN[4] merupakan CL (*cut length*), yang mana nilainya dapat diambil dari *node Geometry.ConcreteBeamColumnCharacteristics* seperti pada Gambar 3.9.



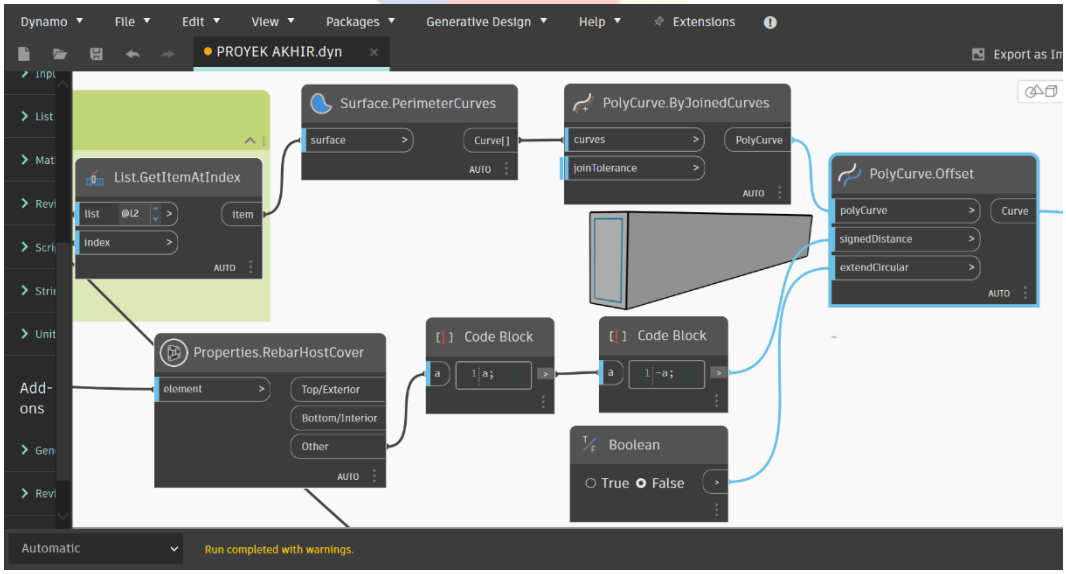

Gambar 3.28. Isi skrip Python pada *node Calc*

Pada Gambar 3.28, digunakan komponen kondisi pada Python. Salah satu contohnya dapat dilihat pada bagian "longi_polos". Pada bagian tersebut terdapat pengondisian terhadap persamaan panjang penyaluran (ld) sebagaimana seperti pada Tabel 2.3. Setelah itu, akan dihitung panjang sambungan (lst). Untuk "longi_polos", panjang total (PT) merupakan penjumlahan antara CL, ld, dan lst. *Output* dari *node Calc* adalah PT, sehingga nantinya *node* akan menunjukkan hasil seperti pada Gambar 3.29.

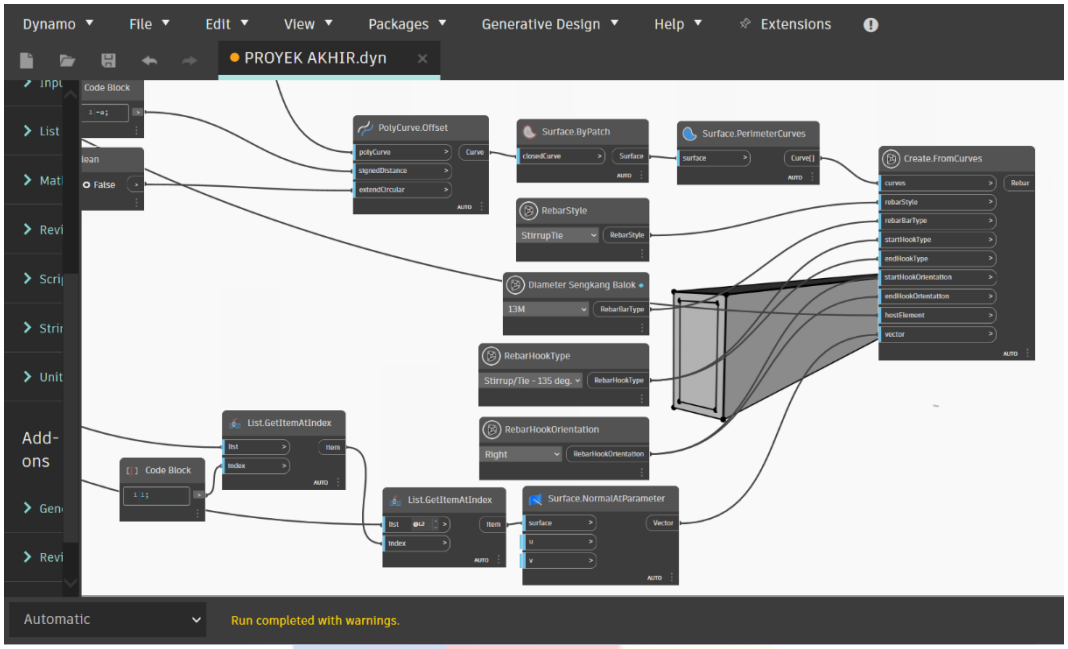



Gambar 3.29. *Output* dari *node Calc* yang ditunjukkan oleh *node Watch*

Dalam membuat tulangan sengkang, terdapat beberapa perbedaan langkah dengan longitudinal. Gambar 3.30 menunjukkan adanya perbedaan *nodes* dengan Gambar 3.12 *Nodes PolyCurve.ByJoinedCurves* berperan dalam mengubah kurva-kurva yang telah dibentuk oleh *node Surface.PerimeterCurves* menjadi kurva yang menerus. Oleh karena itu, dalam membuat garis *offset* selimut beton, maka *output* tadi harus dijadikan *input* pada *node PolyCurve.Offset*.

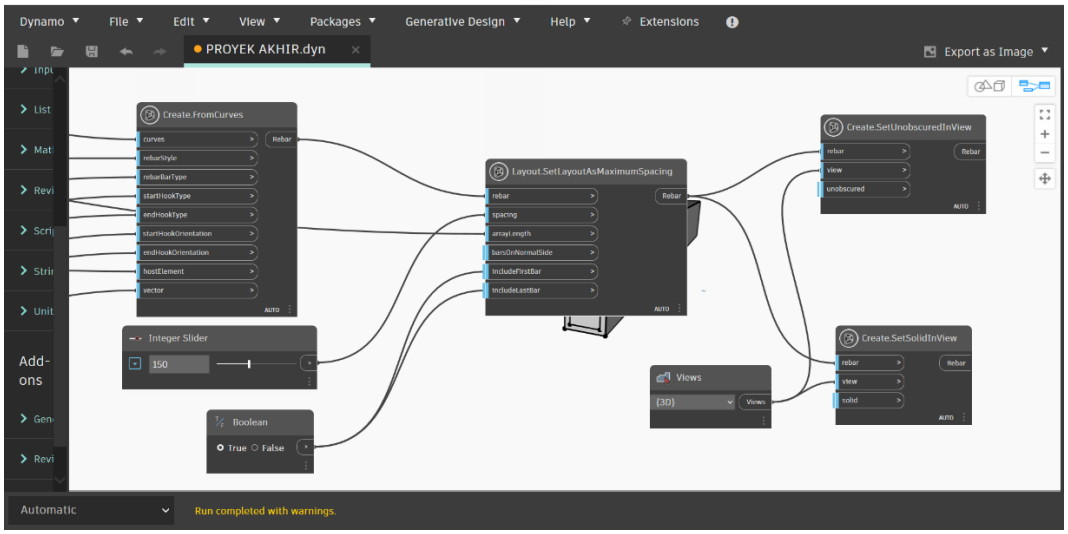



Gambar 3.30. Menetapkan titik awal untuk tulangan sengkang



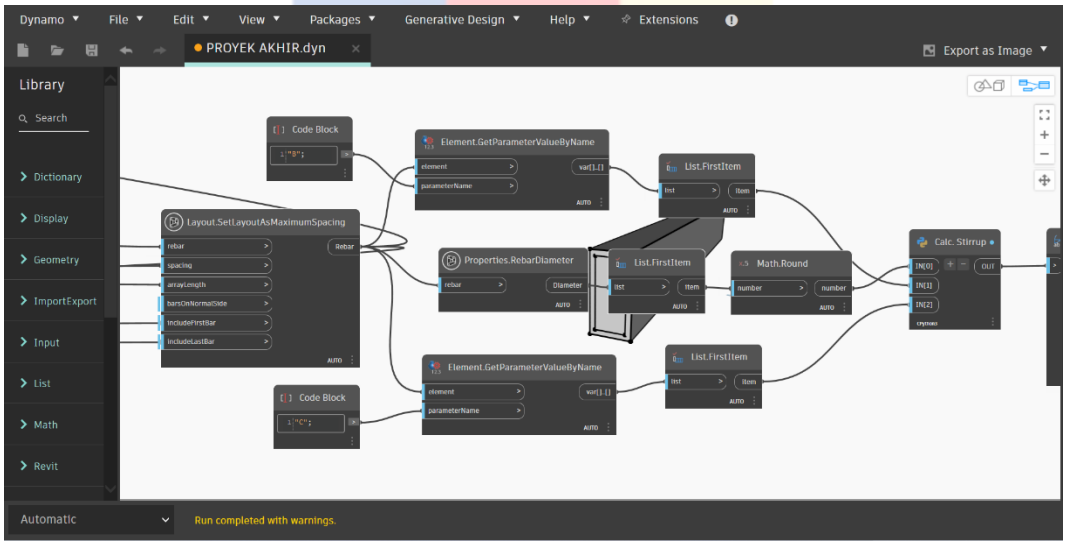

Gambar 3.31. Membuat tulangan sengkang pada balok

*Node Surface.ByPatch* seperti pada Gambar 3.31 berperan dalam mengisi area di dalam garis *offset* yang dibuat oleh *node PolyCurve.Offset*, sehingga sengkang akan terisi di dalam area tersebut. Kemudian area permukaan yang dibentuk oleh *node PolyCurve.Offset* harus diterjemahkan ke bentuk kurva sebagai *input* bagi *node Create.FromCurves*. Oleh karena itu, harus digunakan *node Surface.PerimeterCurves*. Kemudian untuk *input vector* pada *node Create.FromCurves,* maka sengkang harus mengarah ke ujung bentang balok, sehingga lintasan sengkang adalah sepanjang bentang balok.




Gambar 3.32. Menentukan jumlah sengkang

Gambar 3.32 menunjukkan cara menentukan jumlah sengkang. Caranya dengan menggunakan *node Layout.SetLayoutAsMaximumSpacing*, dan jarak spasi antarsengkang dapat diatur dengan *node Integer Slider*. Lalu *input arrayLength* adalah *CutLength*, sehingga sengkang akan terpasang pada jarak *cut length* balok. *Nodes IncludeFirstBar* dan *IncludeLastBar* merupakan *node* yang mengatur apakah sengkang pertama dan sengkang terakhir ingin ditampilkan pada Revit. *Input* dari kedua *nodes* tersebut menggunakan *node Boolean*. Apabila tidak ingin disertakan sengkang pertama dan sengkang akhir, maka cukup memilih *False* pada *node Boolean*. Setelah *node Layout.SetLayoutAsMaximumSpacing* berhasil membuat tulangan sengkang, maka hasil tersebut harus dihubungkan dengan *nodes Create.SetUnobscuredInView* dan *Create.SetSolidInView* supaya terlihat tampilannya pada Revit.

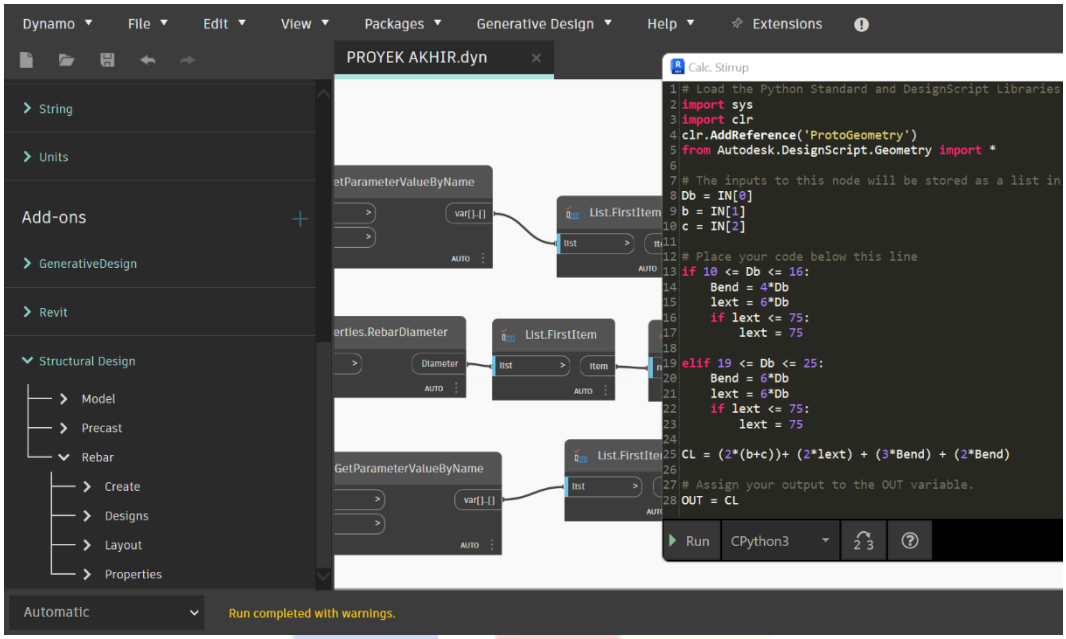



Gambar 3.33. *Input* pada *node Calc. Stirrup*

Gambar 3.33 menunjukkan urutan langkah program dalam menghitung panjang sengkang. *Node Calc. Stirrup* memiliki *inputs* berupa:

- IN[0] merupakan diameter tulangan, yang mana diameter tersebut akan mengikuti ketentuan dari Tabel 2.6 dalam menentukan diameter sisi dalam bengkokan (*bend*) dan perpanjangan kait ($l_{ext}$);
- IN[1] merupakan nilai dimensi B (lebar) dari tulangan sengkang; dan
- IN[2] merupakan nilai dimensi C (tinggi) dari tulangan sengkang.

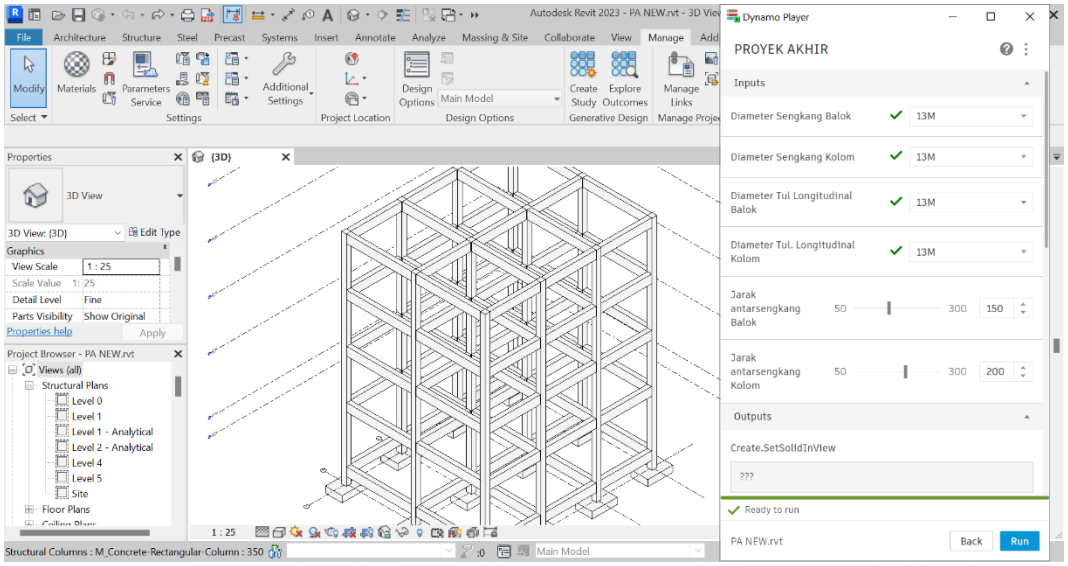
Ketiga *inputs* tersebut akan diolah oleh program untuk dihitung panjang aktual satu sengkang (CL), menggunakan persamaan seperti pada Gambar 3.34. Panjang sengkang merupakan penjumlahan dari keliling sengkang, 2 kali perpanjangan kait sengkang ($l_{ext}$), 3 kali bengkokan sisi dalam (*bend*) 90°, dan 2 kali bengkokan sisi dalam (*bend*) 135°.



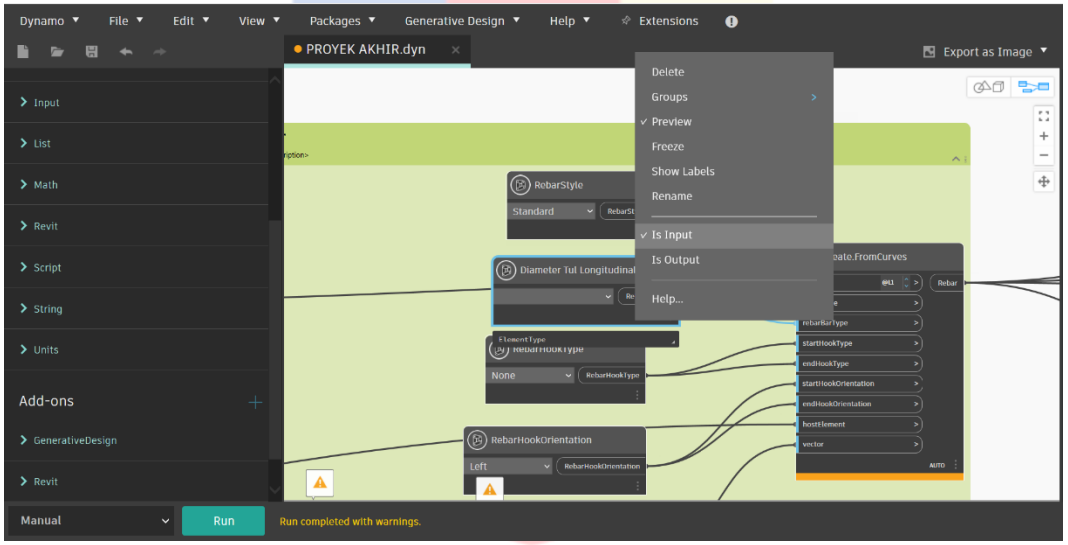

Gambar 3.34. Skrip Python pada *node Calc. Stirrup*

Salah satu keunggulan Dynamo yang mempermudah penggunaan program adalah fitur Dynamo Player. Fitur ini memungkinkan pengguna untuk menjalankan program tanpa harus membuka halaman pemrograman. Gambar 3.35 memperlihatkan bagaimana tampilan dari Dynamo Player yang lebih sederhana dari tampilan program. Namun, tetap dapat menjalankan program secara keseluruhan.

Sejatinya, Dynamo Player terhubung dengan program yang telah dibuat. Hal ini dapat dilihat pada *Inputs* dan *Outputs* pada tampilan Dynamo Player, yang mana *Inputs* dan *Outputs* tersebut merupakan *nodes* yang ada pada program. Untuk mengubah *nodes* pada program sebagai *Inputs* atau *Outputs* pada Dynamo Player, maka harus dilakukan klik kanan pada *node* program yang diinginkan, kemudian menentukan *node* tersebut akan menjadi *Input* atau *Output*. Hal ini dapat dilihat pada Gambar 3.36.



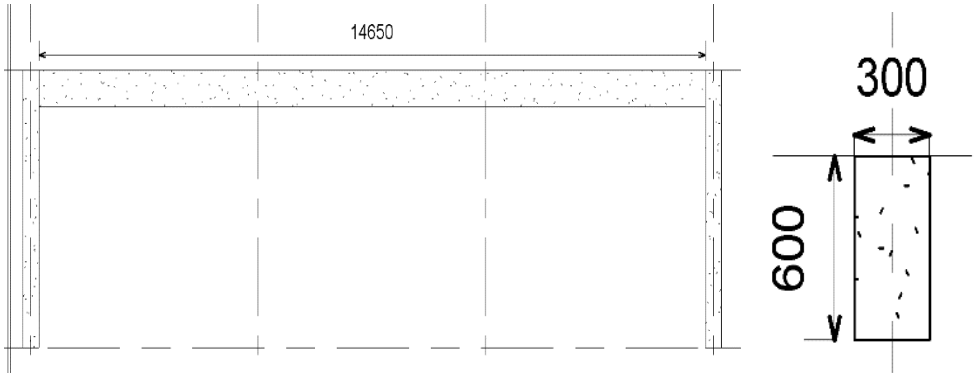

Gambar 3.35. Tampilan dari Dynamo Player




Gambar 3.36. Cara mengatur *node* sebagai *input* pada Dynamo Player

## III.6. Validasi Program

Validasi program dilakukan untuk memeriksa apakah hasil perhitungan pada program sudah sesuai dengan hasil hitungan secara manual. Terkadang penulisan kode pemrograman dapat dibaca secara lain oleh program (ambiguitas), sehingga memengaruhi hasil perhitungan program. Oleh karena itu, penulisan kode program yang menghitung nilai panjang tulangan harus selalu diperiksa untuk memastikan bahwa kode tersebut telah ditulis dengan urutan dan cara yang benar, sehingga siap untuk digunakan pada studi kasus.




Gambar 3.37. Contoh struktur balok untuk validasi

Validasi program akan menghitung panjang tulangan longitudinal pada struktur balok sederhana dengan bentang 14,65 m, dengan ukuran 300 x 600 mm seperti pada Gambar 3.37. Diasumsikan bahwa balok akan menggunakan tulangan 4D16 dengan kait di kedua sisi. Selain itu, untuk asumsi faktor pada persamaan adalah sebagai berikut.

- $\lambda$ = 1
- $f'_c$ = 28 MPa
- $\psi_c$ = 1
- $f_y$ = 420 MPa
- $\psi_e$ = 1
- $\psi_r$ = 1

Untuk perhitungan panjang tulangan secara manual adalah sebagai berikut.

$$l_{dh} = \left[\left(\frac{0{,}24 \,.\, f_y \,.\, \psi_e \,.\, \psi_c \,.\, \psi_r}{\lambda \,.\, \sqrt{f'_c}}\right) . d_b\right] . 2$$

$$l_{dh} = \left[\left(\frac{0{,}24 \,.\, 420 \,.\, 1 \,.\, 1 \,.\, 1}{1 \,.\, \sqrt{28}}\right) . 16\right] . 2$$

$$\underline{l_{dh} = 609{,}58\ mm}$$

$$Bend = 6d_b$$

$$Bend = 6 \,.\, 16$$

$$\underline{Bend = 96\ mm}$$

$$l_{ext} = 12 d_b$$

$$l_{ext} = 12 \,.\, 16$$

$$\underline{l_{ext} = 192\ mm}$$

Untuk panjang bersih $l_{dh}$ merupakan pengurangan antara $l_{dh}$ dengan jari-jari bengkokan. Oleh karena itu, panjang bersih $l_{dh}$ adalah:

$$l_{dh}\ bersih = l_{dh} - [(\frac{Bend}{2}) + d_b]$$

$$l_{dh}\ bersih = 609{,}58 - [(\frac{96}{2}) + 16]$$

$$\underline{l_{dh}\ bersih = 481{,}58\ mm}$$

Selanjutnya, perhitungan panjang total tulangan longitudinal adalah dengan menjumlahkan $l_{dh}$ bersih, $l_{ext}$, dan panjang bengkokan. Panjang bengkokan dihitung dengan persamaan:

$$Panjang\ bengkokan = 0{,}25\,.\,2\,.\,\pi.\,(\frac{Bend}{2} + \frac{d_b}{2})$$

$$Panjang\ bengkokan = 0{,}25\,.\,2\,.\,\pi.\,(\frac{96}{2} + \frac{16}{2})$$

$$\underline{Panjang\ bengkokan = 88\ mm}$$

Oleh karena bentang balok adalah 15 m, sehingga diperlukan adanya sambungan pada tulangan, sehingga panjang lewatan ($l_{st}$) dihitung dengan cara:

$$l_{st} = 1.3 * l_{dh}$$

$$l_{st} = 1.3 * 609{,}58$$

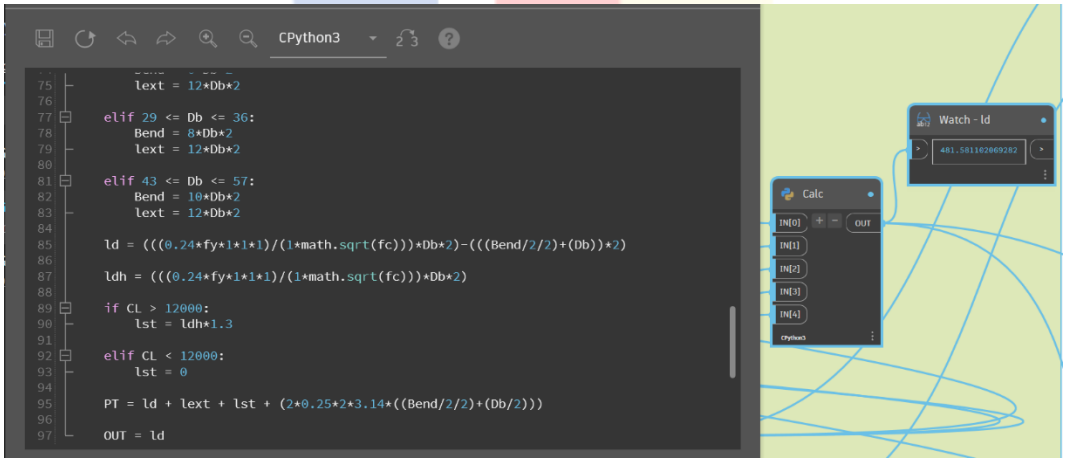
$$\underline{l_{st} = 792{,}455\ mm}$$

$$Panjang\ total\ tulangan\ = (14.650 + 481{,}58 + (2\,.\,192) + (2\,.\,88) + 792{,}455).\,4$$

$$Panjang\ total\ tulangan\ = (16.483{,}87).\,4$$

$$\underline{Panjang\ total\ tulangan = 65.936\ mm\ \approx 65{,}94\ m}$$

Untuk validasi pada program adalah sebagai berikut.

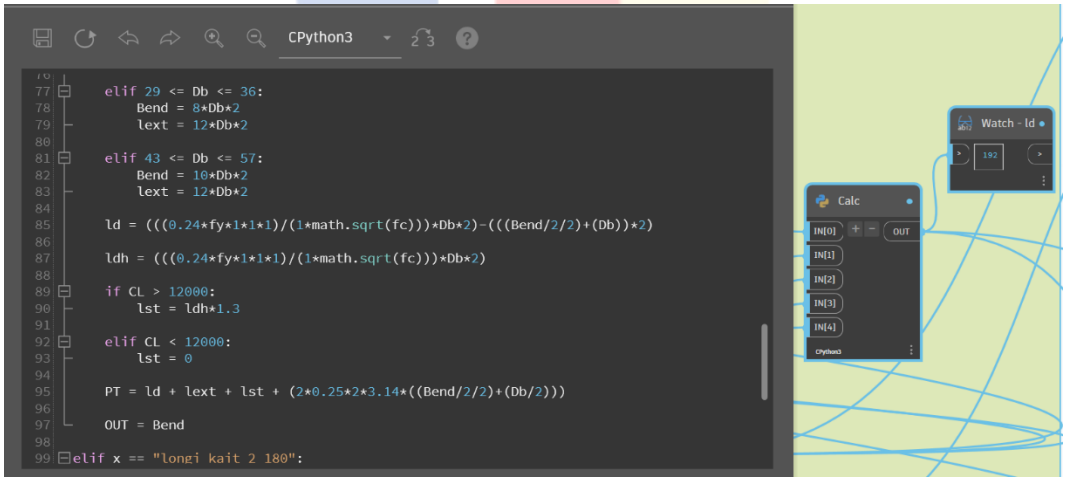



Gambar 3.38. Validasi program saat menghitung $l_d$

Gambar 3.38 menunjukkan bahwa kode pemrograman dalam menghitung $l_d$ telah direduksi dengan faktor bengkokan, sehingga $l_d$ yang dihitung oleh program merupakan $l_d$ bersih. Hasil perhitungan program pada $l_d$ bersih adalah 481,58 mm. Hasil tersebut telah sama dengan hasil perhitungan manual.


Gambar 3.39. Validasi program saat menghitung $l_{ext}$

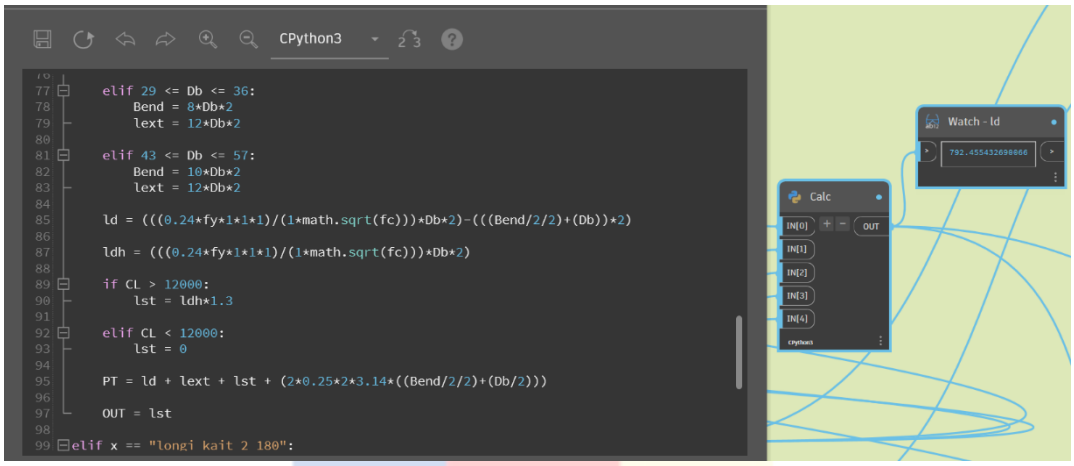
Gambar 3.39 menunjukkan bahwa kode penulisan program untuk menghitung $l_{ext}$ dikalikan dengan 2. Hal ini menunjukkan bahwa ketika tulangan dibaca oleh program sebagai tulangan yang menggunakan kait 2 sisi, maka panjang $l_{ext}$ dan *Bend* akan dikalikan 2. Hasil menunjukkan bahwa panjang $l_{ext}$ pada program adalah 384 mm, sedangkan pada hitungan manual adalah 192 mm. Namun dalam perhitungan manual, ketika akan menghitung panjang total tulangan, maka $l_{ext}$ akan dikalikan dengan 2 karena kait yang terdapat pada satu tulangan logitudinal ada 2.




Gambar 3.40. Validasi program saat menghitung *Bend*

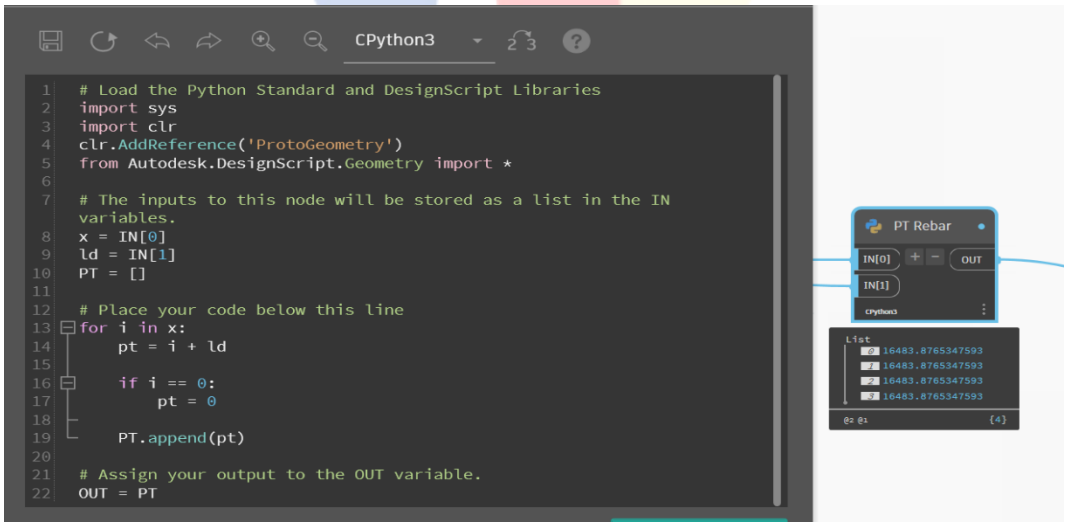
Gambar 3.40 menunjukkan bahwa kode penulisan program untuk menghitung *Bend* juga dikalikan dengan 2. Hal ini menunjukkan bahwa ketika tulangan dibaca oleh program sebagai tulangan yang menggunakan kait 2 sisi, maka panjang $l_{ext}$ dan *Bend* akan dikalikan 2. Hasil menunjukkan bahwa panjang *Bend*

pada program adalah 192 mm, sedangkan pada hitungan manual adalah 96 mm. Ketika akan menghitung panjang total tulangan, maka *Bend* akan dikalikan dengan 2 sehingga nilainya akan sama dengan perhitungan program.




Gambar 3.41. Validasi program saat menghitung $l_{st}$

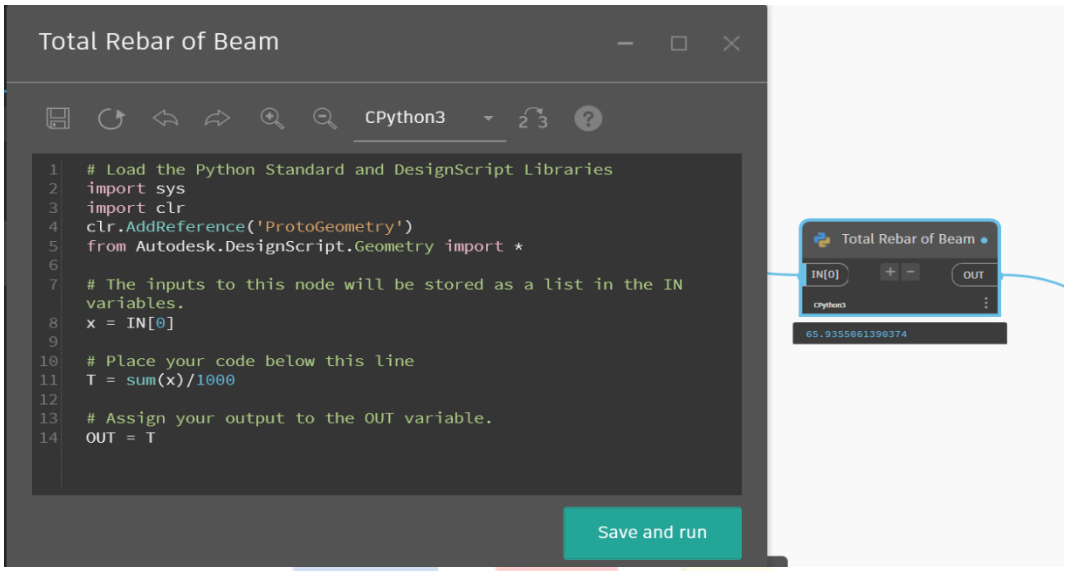
Gambar 3.41 menunjukkan bahwa kode penulisan program untuk menghitung $l_{st}$. Panjang $l_{st}$ dihitung menggunakan panjang $l_{dh}$ pada program yang merupakan panjang lewatan kotor. Pada program, $l_{st}$ dihitung dengan mengalikan 1,3 dengan $l_{dh}$, dan program menghitung $l_{st}$ sebesar 792,455 mm. Hasil tersebut sama dengan hasil pada perhitungan manual.




Gambar 3.42. Validasi program saat menghitung panjang tiap tulangan

Gambar 3.42 menunjukkan program mendeteksi bahwa terdapat 4 tulangan longitudinal pada balok. Setelah semua panjang pada *node* sebelumnya

dijumlahkan dengan panjang bentang bersih balok, maka program menghitung panjang tersebut sebesar 16.483,87 mm. Hasil tersebut telah sesuai dengan hasil perhitungan manual. Langkah selanjutnya dari hasil tersebut akan menjadi *input* untuk menghitung total panjang tulangan longitudinal.

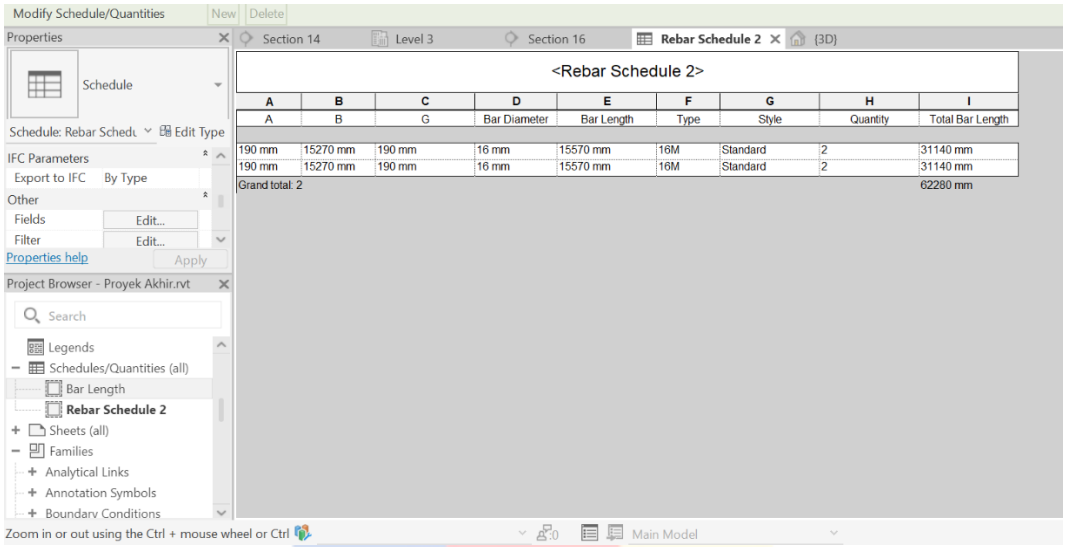



Gambar 3.43. Validasi program saat menghitung total tulangan longitudinal

Gambar 3.43 menunjukkan bahwa penjumlahan masing-masing panjang tulangan yang terdapat pada Gambar 3.42, maka hasilnya adalah 65.935,50 mm. Namun, jika diperhatikan pada kode pemrogramannya, hasil penjumlahan tersebut dibagi dengan 1.000 yang menunjukkan bahwa hasil perhitungan akan ditampilkan dalam satuan meter. Oleh karena itu, hasil perhitungan total tulangan longitudinal adalah 65.935,50 mm ≈ 65,94 m. Hasil tersebut telah sesuai dengan hasil perhitungan manual.

Hasil perhitungan antara program dan manual telah sesuai. Setelah itu, akan dibandingkan hasil perhitungan antara program dengan Revit biasa. Pada Revit biasa, tulangan dimodelkan secara manual sesuai dengan contoh. Pada Gambar 3.44, dapat dilihat bahwa panjang total tulangan yang dihitung oleh Revit adalah 62.280 mm ≈ 62,28 m. Hasil tersebut memiliki selisih sebesar 3,66 m. Pada Gambar 3.45 dapat dilihat bahwa pemodelan pada Revit biasa tidak memasukkan *overlapping* secara otomatis, walaupun bentang balok melebihi 12 m. Gambar 3.46 menunjukkan bahwa panjang kait yang dibaca oleh Revit adalah 128,6 mm. Kemudian panjang bentang bersih tulangan lurus yang dibaca kait pada Gambar

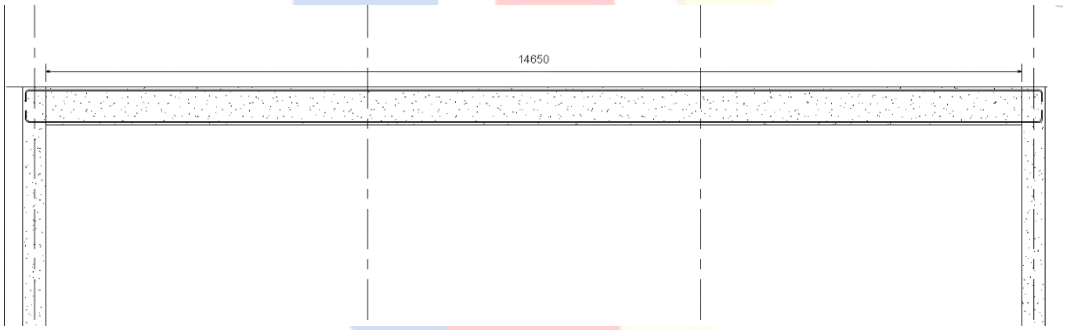
4.47 adalah 15.143,2 mm. Terakhir, panjang bengkokan yang dibaca Revit pada Gambar 4.48 adalah 87 mm.



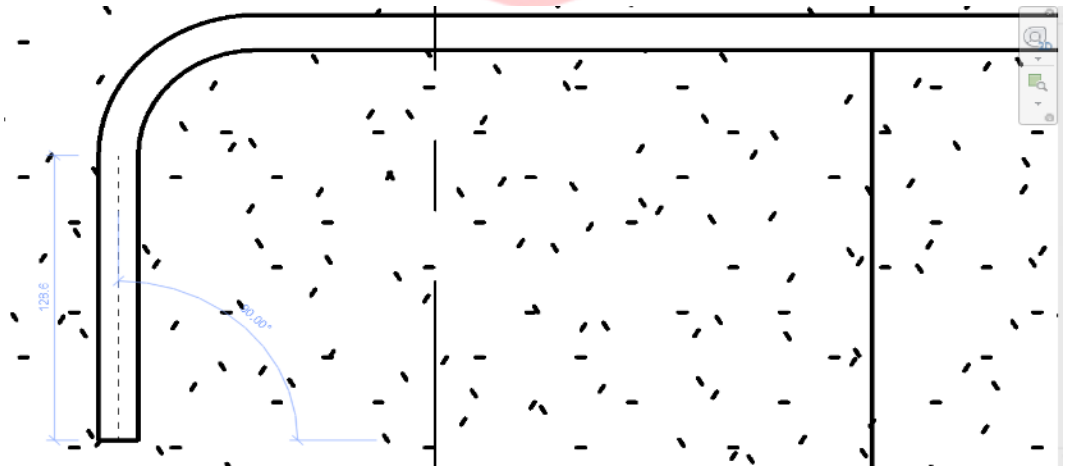

<Rebar Schedule 2>

| A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| A | B | G | Bar Diameter | Bar Length | Type | Style | Quantity | Total Bar Length |
| 190 mm | 15270 mm | 190 mm | 16 mm | 15570 mm | 16M | Standard | 2 | 31140 mm |
| 190 mm | 15270 mm | 190 mm | 16 mm | 15570 mm | 16M | Standard | 2 | 31140 mm |
| Grand total: 2 | | | | | | | | 62280 mm |

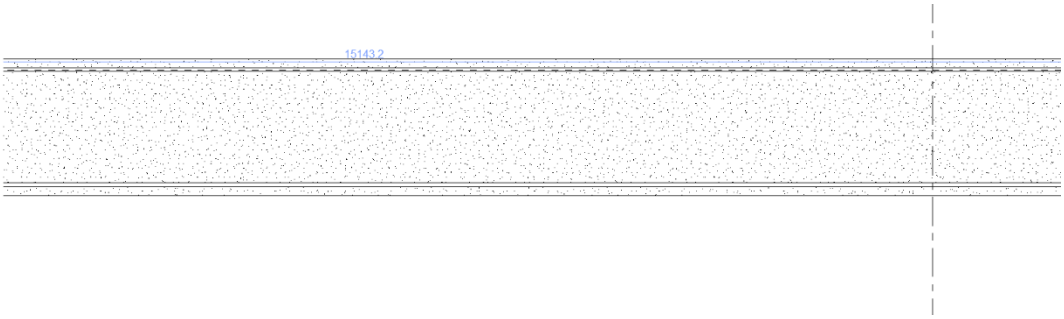
Gambar 3.44. Perhitungan total tulangan pada Revit



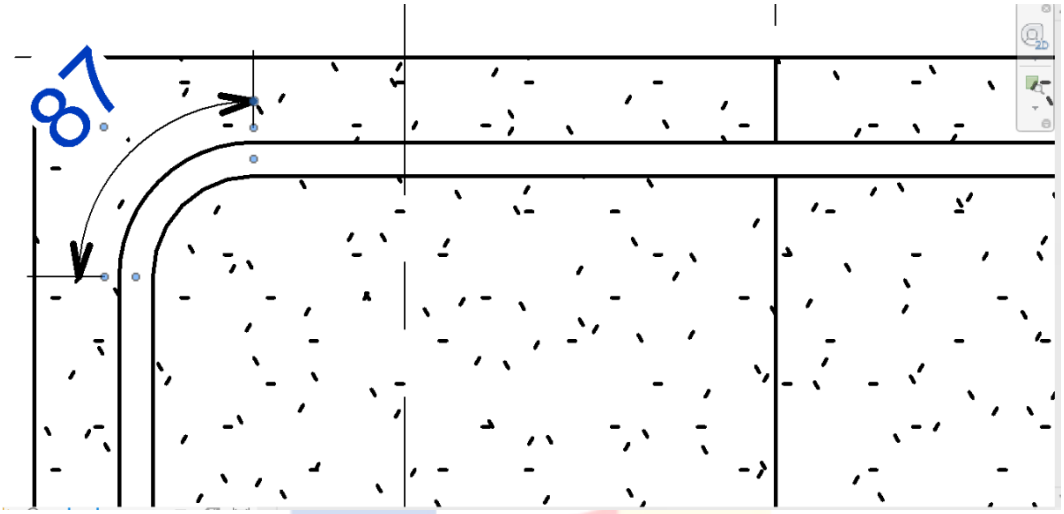

Gambar 3.45. Pemodelan tulangan oleh Revit biasa




Gambar 3.46. Panjang kait pada Revit



Gambar 3.47. Panjang bentang bersih pada Revit




Gambar 3.48. Panjang bengkokan pada Revit

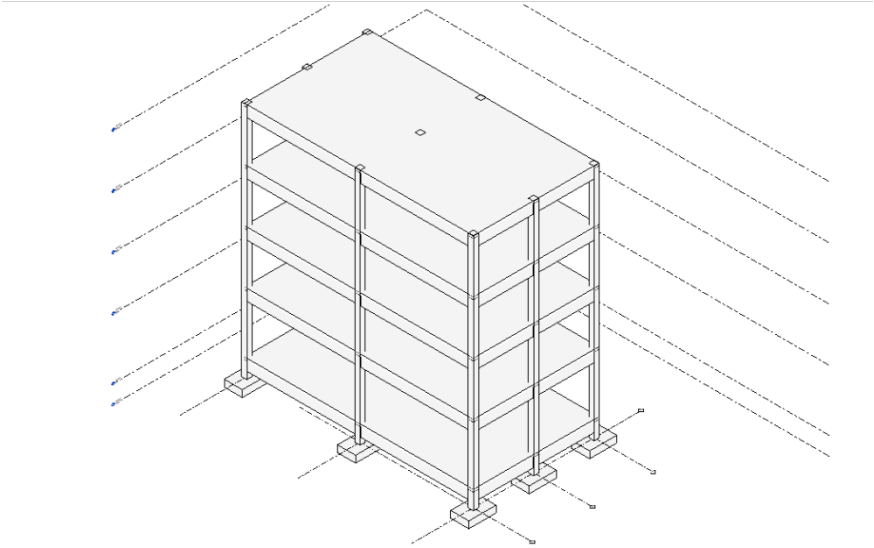
Jika semua panjang yang dibaca oleh Revit dijumlahkan, maka panjang untuk satu tulangan adalah 15.574,4 mm. Namun, Revit membulatkan hitungan tersebut secara otomatis menjadi 15.570 mm. Hasil tersebut lebih kecil dibanding dengan hasil yang dihitung oleh program, yaitu 16.483,87 mm. Kemudian, jika keempat tulangan tersebut dijumlahkan, maka panjang total tulangannya adalah 62.280 mm.

### III.7. Studi Kasus

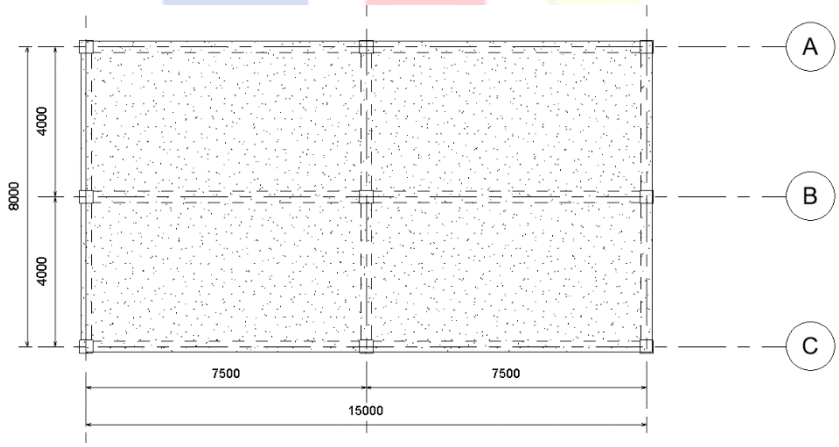
Untuk studi kasus pada Tugas Akhir ini, akan menggunakan struktur gedung hipotetik berupa bangunan beton bertulang 4 lantai, seperti pada Gambar 3.49. Studi kasus merupakan bangunan yang terdiri dari:

- Fondasi *foot plat* sebanyak 9 buah dengan dimensi: 1,8 x 1,2 x 0,45 m;
- Kolom setinggi 15,7 m sebanyak 9 buah dengan dimensi: 350 x 350;
- Balok uk. 300 x 600 dengan bentang 15 m dan 8 m; dan
- Pelat lantai setebal 150 mm.

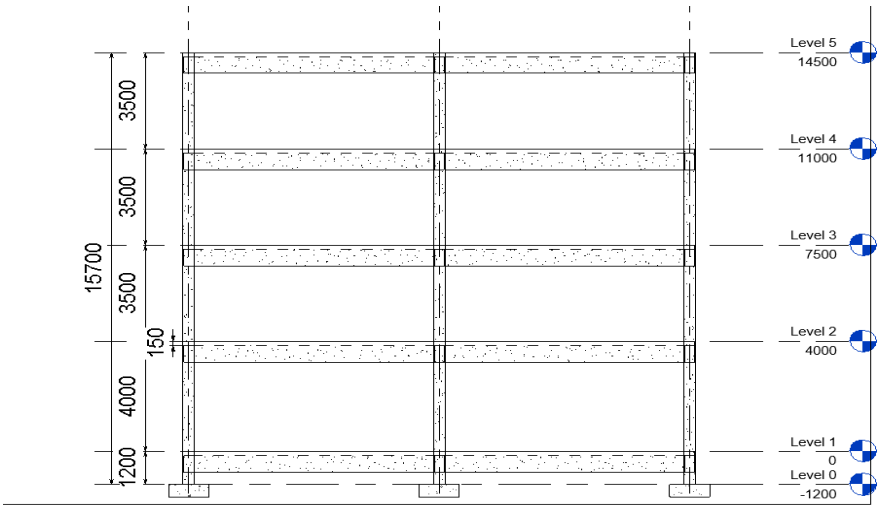
Keterangan mengenai dimensi, tinggi, dan ketebalan elemen struktur dapat dilihat pada Gambar 3.50, Gambar 4.51, Gambar 4.52, Gambar 4.53, dan Gambar 4.54.



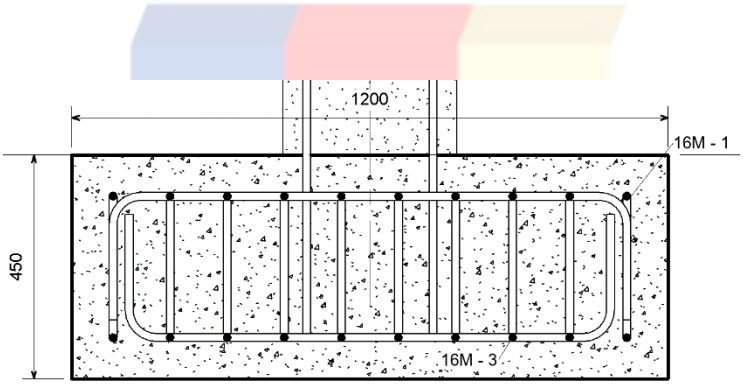
Gambar 3.49. Studi kasus struktur hipotetik gedung 4 lantai



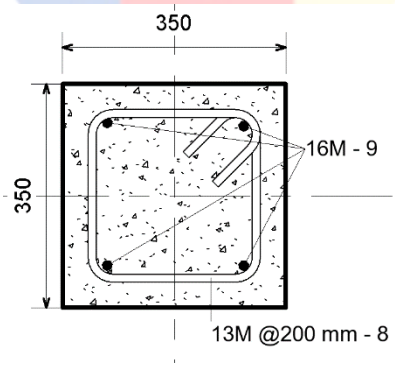

Gambar 3.50. *Floor plan* studi kasus



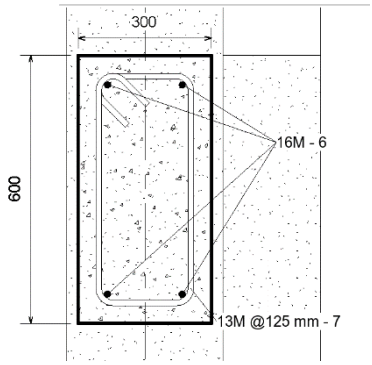

Gambar 3.51. Tinggi antarlantai dan tebal pelat studi kasus




Gambar 3.52. Dimensi penampang fondasi *foot plat* pada studi kasus




Gambar 3.53. Dimensi kolom pada studi kasus




Gambar 3.54. Dimensi penampang balok pada studi kasus

Dari studi kasus tersebut, akan dihitung panjang tulangan dengan dua metode, yaitu metode konvensional dan metode program. Metode konvensional dilakukan dengan menghitung panjang tulangan sesuai peraturan SNI 2847:2019 secara manual menggunakan Microsoft Excel. Sedangkan metode program dilakukan dengan menjalankan program yang telah dibuat pada Dynamo dengan menggunakan bantuan kode Python. Hasil akhir dari perhitungan metode program adalah hasil perhitungan pada program harus sama dengan hasil perhitungan konvensional.